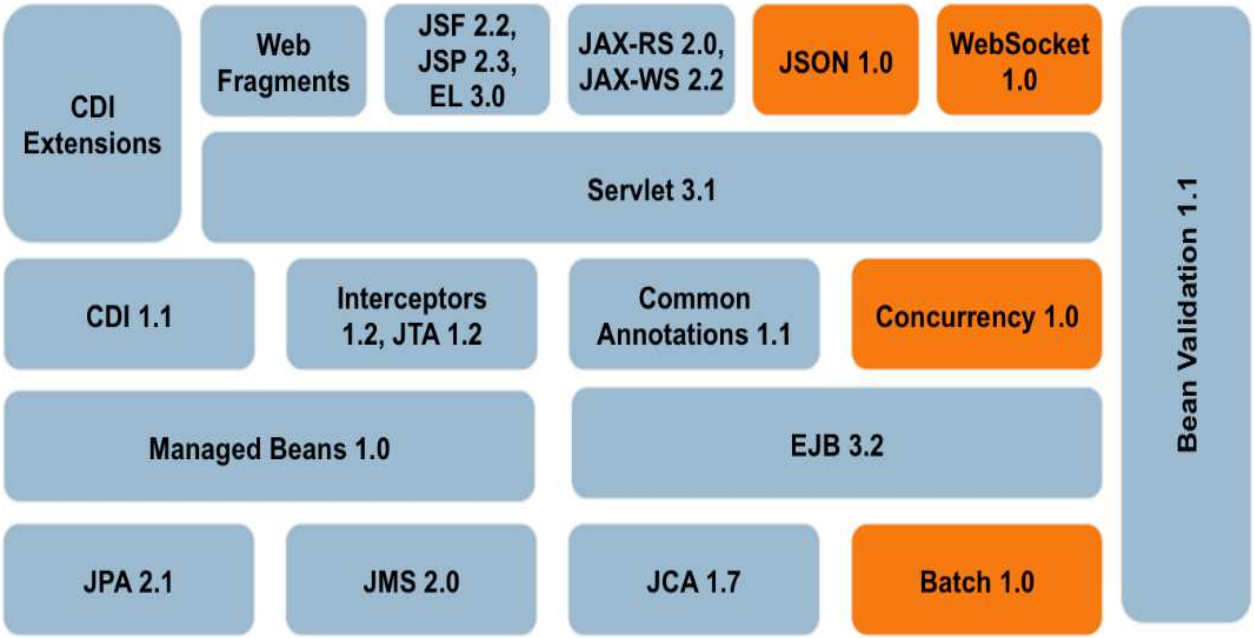
Java Enterprise Edition

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

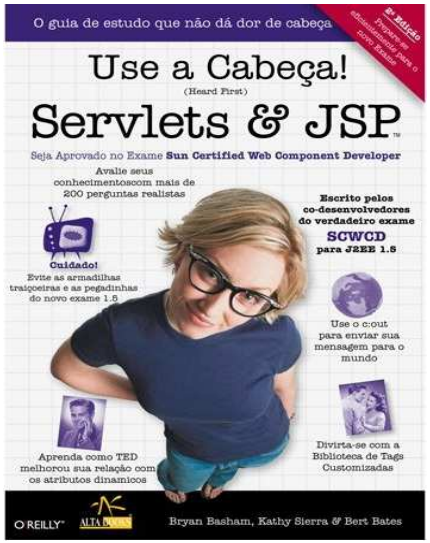
Leonardo Marcelino
http://www.itnerante.com.br/profile/LeonardoMarcelino

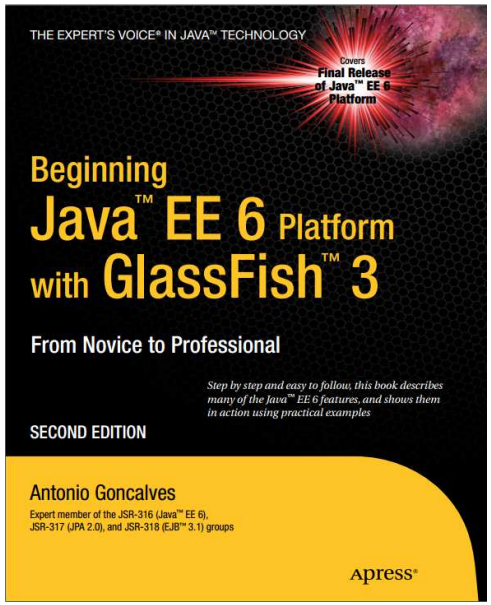
Java Enterprise Edition

- Módulo 2
  - ✓ APIs da plataforma JavaEE
    - Classificação
    - Servlets



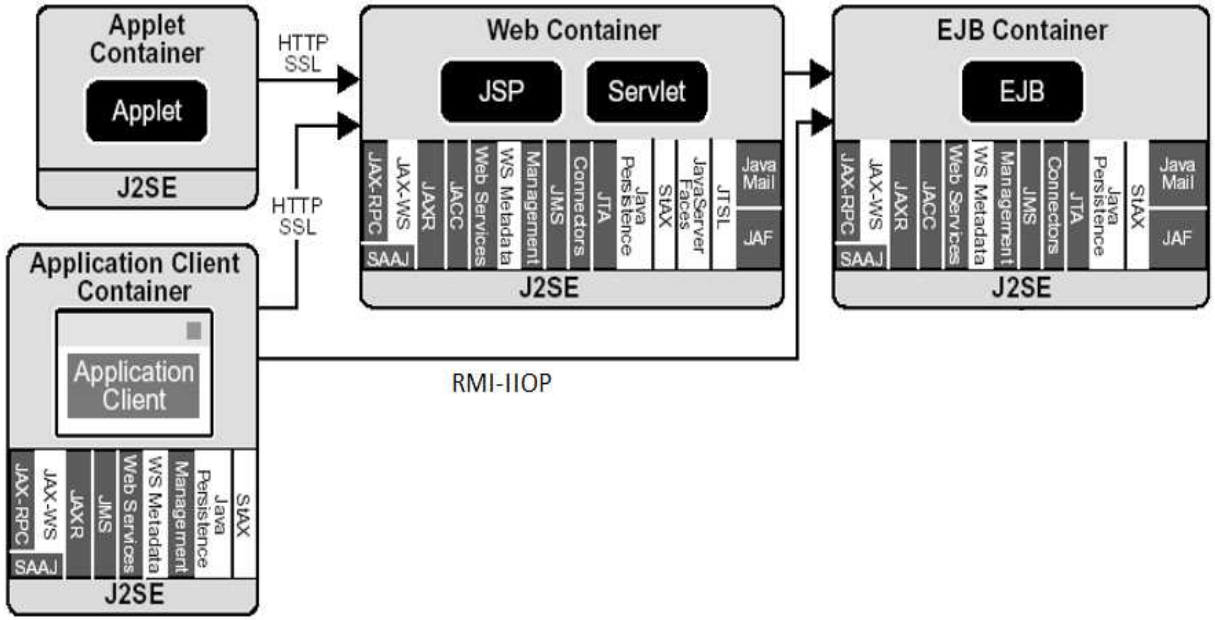
REFERÊNCIAS





Java Enterprise Edition

- API's por container
  - APIs multicontainer
  - containers
    - Application Client
    - Applet
    - Web
    - EJB



- API's por tecnologia
  - a partir da JEE6
  - categorias
    - Java EE-related Specs in Java SE
    - Web Application Technologies
    - Web Services Technologies
    - Enterprise Application Technologies
    - Management and Security Technologies

API's por container

Java EE 5	Web	EJB	App Client
Annotations	X	X	X
Connectors	X	X	
EJB	X	X	X
EL	X		
JACC	X	X	
JAF	X	X	X
Java Persistence	X	X	X
JavaMail	X	X	X
JAXB	X	X	X
JAXR	X	X	X
JAX-RPC	X	X	X
JAX-WS	X	X	X
JMS	X	X	X
JSF	X		
JSP	X		
JSTL	X		
JTA	X	X	
Management	X	X	X
SAAJ	X	X	X
Servlet	X		
StAX	X	X	X
Web Services	X	X	X
WS Metadata	X	X	X

Java EE 6	Web	EJB	App Client
Bean Validation	X	X	X
CDI & DI	X	X	X
EJB Lite	X	X	X
Interceptors	X	X	X
JASPIC	X	X	
JAX-RS	X		
Managed Beans	X	X	X

Java EE 7	Web	EJB	App Client
Concurrency Utilities	X	X	
Batch	X	X	
JSON-P	X	X	X
WebSocket	X		

APIs Java SE	Web	EJB	App Client
	X	X	X

API's por tecnologia

Java EE-related Specs in Java SE

JEE 6	
Java Database Connectivity (JDBC) 4.0	JSR 221
Java API for XML Processing (JAXP) 1.3	JSR 206
JavaBeans Activation Framework (JAF) 1.1	JSR 925
Java Management Extensions (JMX) 2.0	JSR 225 / JSR 003
Streaming API for XML (StAX) 1.0	JSR 173
Java Naming and Directory Interface (JNDI)	
Java Authentication and Authorization Service (JAAS)	
JEE 7	
Java Database Connectivity (JDBC) 4.0	JSR 221
Java API for XML Processing (JAXP) 1.3	JSR 206
JavaBeans Activation Framework (JAF) 1.1	JSR 925
Java Management Extensions (JMX) 2.0	JSR 225 / JSR 003
Streaming API for XML (StAX) 1.0	JSR 173
Java Naming and Directory Interface (JNDI)	
Java Authentication and Authorization Service (JAAS)	
Java Architecture for XML Binding (JAXB) 2.2	JSR 222

API's por tecnologia

Web Application Technologies

JEE 6	
Java Servlet 3.0	JSR 315
JavaServer Pages (JSP) 2.2 Expression Language 2.2	JSR 245
Standard Tag Library for JavaServer Pages (JSTL) 1.2	JSR 52
JavaServer Faces (JSF) 2.0	JSR 314
Debugging Support for Other Languages 1.0	JSR 45
JEE 7	
Java Servlet 3.1	JSR 340
JavaServer Pages (JSP) 2.3	JSR 245
Expression Language 3.0	JSR 341
Standard Tag Library for JavaServer Pages (JSTL) 1.2	JSR 52
JavaServer Faces (JSF) 2.2	JSR 344
Java API for WebSocket	JSR 356
Java API for JSON Processing (JSON-P)	JSR 353

API's por tecnologia

Web Services Technologies

JEE 6	
Java API for RESTful Web Services (JAX-RS)	JSR 311
Implementing Enterprise Web Services	JSR 109
Java API for XML-Based Web Services (JAX-WS)	JSR 224
Java Architecture for XML Binding (JAXB)	JSR 222
Web Services Metadata for the Java Platform	JSR 181
Java API for XML-Based RPC (JAX-RPC)	JSR 101
Java APIs for XML Messaging SOAP with Attachments API for Java (SAAJ)	JSR 67
Java API for XML Registries (JAXR)	JSR 93
JEE 7	
Java API for RESTful Web Services (JAX-RS) 2.0	JSR 339
Implementing Enterprise Web Services 1.3	JSR 109
Java API for XML-Based Web Services (JAX-WS) 2.2	JSR 224
Web Services Metadata for the Java Platform	JSR 181
Java API for XML-Based RPC (JAX-RPC) 1.1	JSR 101
Java APIs for XML Messaging 1.3 SOAP with Attachments API for Java (SAAJ)	JSR 67
Java API for XML Registries (JAXR) 1.0	JSR 93

API's por tecnologia

Enterprise Application Technologies

JEE 6	
Contexts and Dependency Injection for Java (CDI) 1.0	JSR 299
Dependency Injection for Java 1.0	JSR 330
Bean Validation 1.0	JSR 303
Enterprise JavaBeans 3.1 (includes Interceptors 1.1)	JSR 318
Java EE Connector Architecture 1.6	JSR 322
Java Persistence (JPA) 2.0	JSR 317
Common Annotations for the Java Platform 1.1	JSR 250
Java Message Service API (JMS) 1.1	JSR 914
Java Transaction API (JTA) 1.1	JSR 907
JavaMail 1.4	JSR 919
JEE 7	
Contexts and Dependency Injection for Java (CDI) 1.1	JSR 346
Dependency Injection for Java 1.0	JSR 330
Bean Validation 1.1	JSR 349
Enterprise JavaBeans 3.2	JSR 345
Interceptors 1.2	JSR 318
Java EE Connector Architecture 1.7	JSR 322
Java Persistence (JPA) 2.1	JSR 338
Common Annotations for the Java Platform 1.2	JSR 250
Java Message Service API (JMS) 2.0	JSR 343
Java Transaction API (JTA) 1.2	JSR 907
JavaMail 1.5	JSR 919
Batch Applications for the Java Platform	JSR 352
Concurrency Utilities for Java EE 1.0	JSR 236

API's por tecnologia

Management and Security Technologies

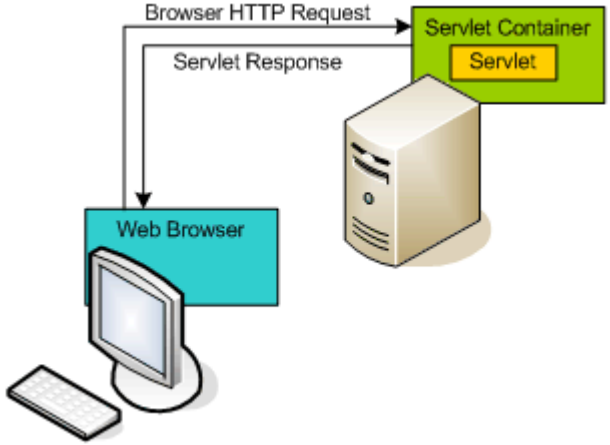
JEE 6	
Java Authentication Service Provider Interface for Containers (JASPIC)	JSR 196
Java Authorization Contract for Containers 1.3 (JACC)	JSR 115
Java EE Application Deployment 1.2	JSR 88
J2EE Management 1.1	JSR 77
JEE 7	
Java Authentication Service Provider Interface for Containers (JASPIC)	JSR 196
Java Authorization Contract for Containers 1.3 (JACC)	JSR 115
Java EE Application Deployment 1.2	JSR 88
J2EE Management 1.1	JSR 77
Debugging Support for Other Languages 1.0	JSR 45

Web Application Technologies

- rodam no lado servidor
  - ✓ container web
  - ✓ processam requisições clientes
  - ✓ lidam com componentes visuais (apresentação)

- APIs
  - ⇨ Java Servlet
  - ⇨ JavaServer Pages (JSP)
  - ⇨ Expression Language
  - ⇨ JavaServer Pages for Standard Tag Library (JSTL)
  - ⇨ JavaServer Faces (JSF)
  - ⇨ Java API for WebSocket
  - ⇨ Java API for JSON Processing (JSON-P)
- web profile
  - ✓ Servlet, JSP, EL, JSTL e JSF
- Tomcat
  - ✓ Servlet e JSP

Servlets



- rodam no lado servidor
  - ✓ container web
  - ✓ processam requisições clientes
  - ✓ lidam com componentes visuais (apresentação)
- Definição

  classe Java usada para estender as capacidades dos servidores que hospedam aplicações acessadas por meio de um modelo de requisição-resposta

  - ✓ respondem qualquer tipo de requisição
  - ✓ mais comum: requisições HTTP
  - ✓ primeira forma de criar aplicações web em Java (página web dinâmicas)
  - ✓ base para aplicações java web
- Estrutura
  - ✓ pacote `javax.servlet`
    - classes e interfaces genéricas

      ⇒ **interface** `Servlet`
      - deve ser implementada por qualquer servlet
      - define os métodos do ciclo de vida

Servlets

- pacote `javax.servlet`
  - ⇒ **interfaces** `ServletRequest e ServletResponse`
    - objetos usados em requisições/respostas genéricas
    - criados pelo container e passados para o servlet
  - ⇒ **classe abstrata** `GenericServlet`
    - servlet independente de protocolo
    - define métodos básicos para servlets
  - ⇒ **interface** `ServletConfig`
    - define métodos para configurar o servlet
    - parâmetros para **UM** servlet durante a inicialização
  - ⇒ **interface** `ServletContext`
    - define métodos para se comunicar com o container
    - parâmetros para **TODOS OS** servlets durante a inicialização

Classes
AsyncEvent
HttpConstraintElement
HttpMethodConstraintElement
MultipartConfigElement
ServletContextAttributeEvent
ServletContextEvent
ServletInputStream
ServletOutputStream
ServletRequestAttributeEvent
ServletRequestEvent
ServletRequestWrapper
ServletResponseWrapper
ServletSecurityElement

Interfaces			
AsyncContext	AsyncListener	Filter	FilterChain
FilterConfig	FilterRegistration	FilterRegistration.Dynamic	Registration
Registration.Dynamic	RequestDispatcher	ServletContainerInitializer	ServletContext
ServletContextAttributeListener	ServletContextListener	ServletRegistration	ServletRegistration.Dynamic
ServletRequestAttributeListener	ServletRequestListener	SessionCookieConfig	SingleThreadModel

Servlets

- Estrutura
  - ✓ pacote `javax.servlet.http`
    - classes e interfaces específicas para o protocolo HTTP

  ⇒**classe abstrata** `HttpServlet`
    - servlet específico para atender requisições HTTP (web)
    - estende GenericServlet

  ⇒**interfaces** `HttpServletRequest e HttpServletResponse`
    - objetos usados em requisições/respostas HTTP
    - criados pelo container e passados para o servlet

Interfaces		
HttpServletRequest	HttpServletResponse	HttpSession
HttpSessionActivationListener	HttpSessionAttributeListener	HttpSessionBindingListener
HttpSessionContext	HttpSessionListener	Part

Classes		
Cookie	HttpServlet	HttpServletRequestWrapper
HttpServletResponseWrapper	HttpSessionBindingEvent	HttpSessionEvent
HttpUtils		

[01] CESPE - 2011 - TRE-ES

Julgue o item que se segue, referente a fundamentos de computação e a linguagens de programação.

Um *servlet* é uma classe Java utilizada para ampliar a capacidade de acesso dos servidores a aplicações por meio do modelo requisição-resposta. Embora os *servlets* possam responder a um tipo específico de requisição hospedada em servidores *web*, os *servlets* não respondem a requisições genéricas.

[02] *FCC - 2010 - DPE-SP*

Servlets são projetadas para fornecer aos desenvolvedores uma solução JAVA para criar aplicações web. Para criar Servlets é necessário importar as classes padrão de extensão dos pacotes

a) javax.servlet e javax.servlet.http.

b) javax.servlet e javax.http.servlet.

c) javax.servlet.html e javax.servlet.http.

d) servlet.javax e servlet.javax.http.

e) javax.servlet.smtp e javax.servlet.html.

[03] FMP-RS - 2013 - MPE-AC

No contexto da arquitetura Java Enterprise Edition, _____ são, em termos de estrutura, classes Java especializadas que se assemelham muito à estrutura dos *applets Java*, porém rodando em um servidor *web* e não no cliente.

Assinale a única alternativa que completa corretamente a lacuna acima.

a) *Java ME (Java Micro Edition)*
b) *portlets*
c) *Java Persistence* API (JPA)
d) *Enterprise JavaBeans* (EJB)
e) *servlets*

[04] VUNESP - 2013 - FUNDUNESP

Na plataforma J2EE, a classe ServletRequest define

a) a estrutura do objeto principal do *Servlet*, permitindo que sejam feitas requisições ao *Servlet.*

b) métodos que permitem que o *Servlet* faça requisições de forma assíncrona.

c) métodos que permitem que o *Servlet* faça requisições aos clientes.

d) propriedades que permitem que seja alterado o comportamento do *Servlet*.

e) um objeto que fornecerá informações sobre a requisição feita pelo cliente ao *Servlet*.

[05] FCC - 2011 - TRT1

Em relação às tecnologias *Java*, é INCORRETO afirmar que as *Servlets*

a) deixam para a API utilizada na sua escrita a responsabilidade com o ambiente em que elas serão carregadas e com o protocolo usado no envio e recebimento de informações.

b) fornecem um mecanismo simples e consistente para estender a funcionalidade de um servidor *Web*.

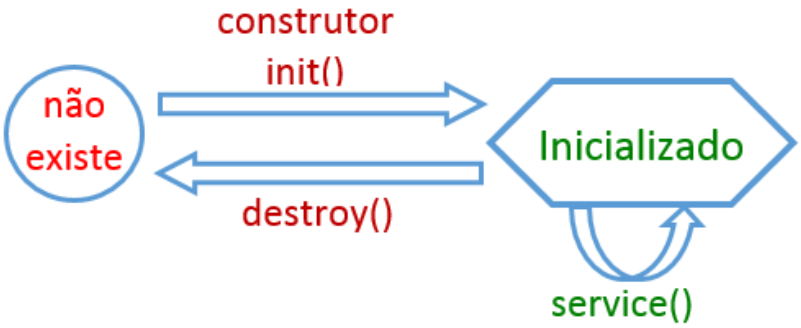
c) podem ser incorporadas em vários servidores *Web* diferentes.

d) podem rodar em qualquer plataforma sem a necessidade de serem reescritas ou compiladas novamente.

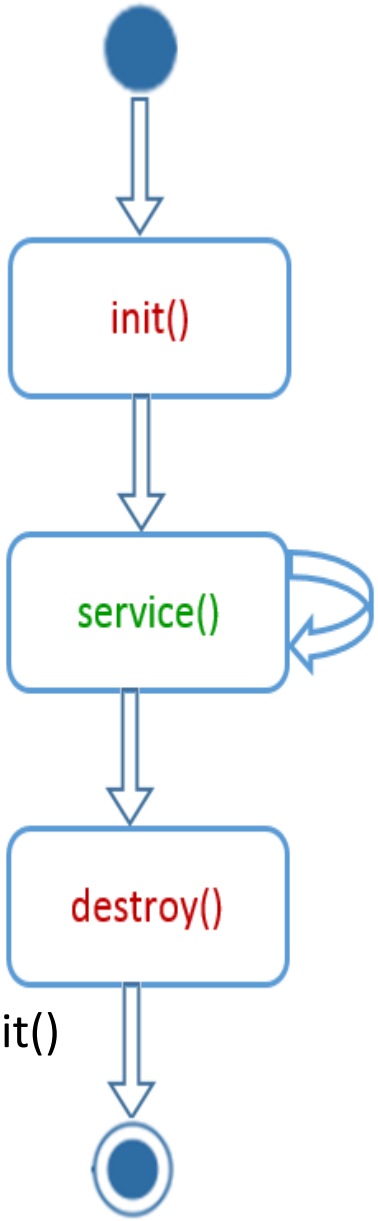
e) são carregadas apenas uma vez e, para cada nova requisição, a *servlet* gera uma nova *thread.*

Servlets

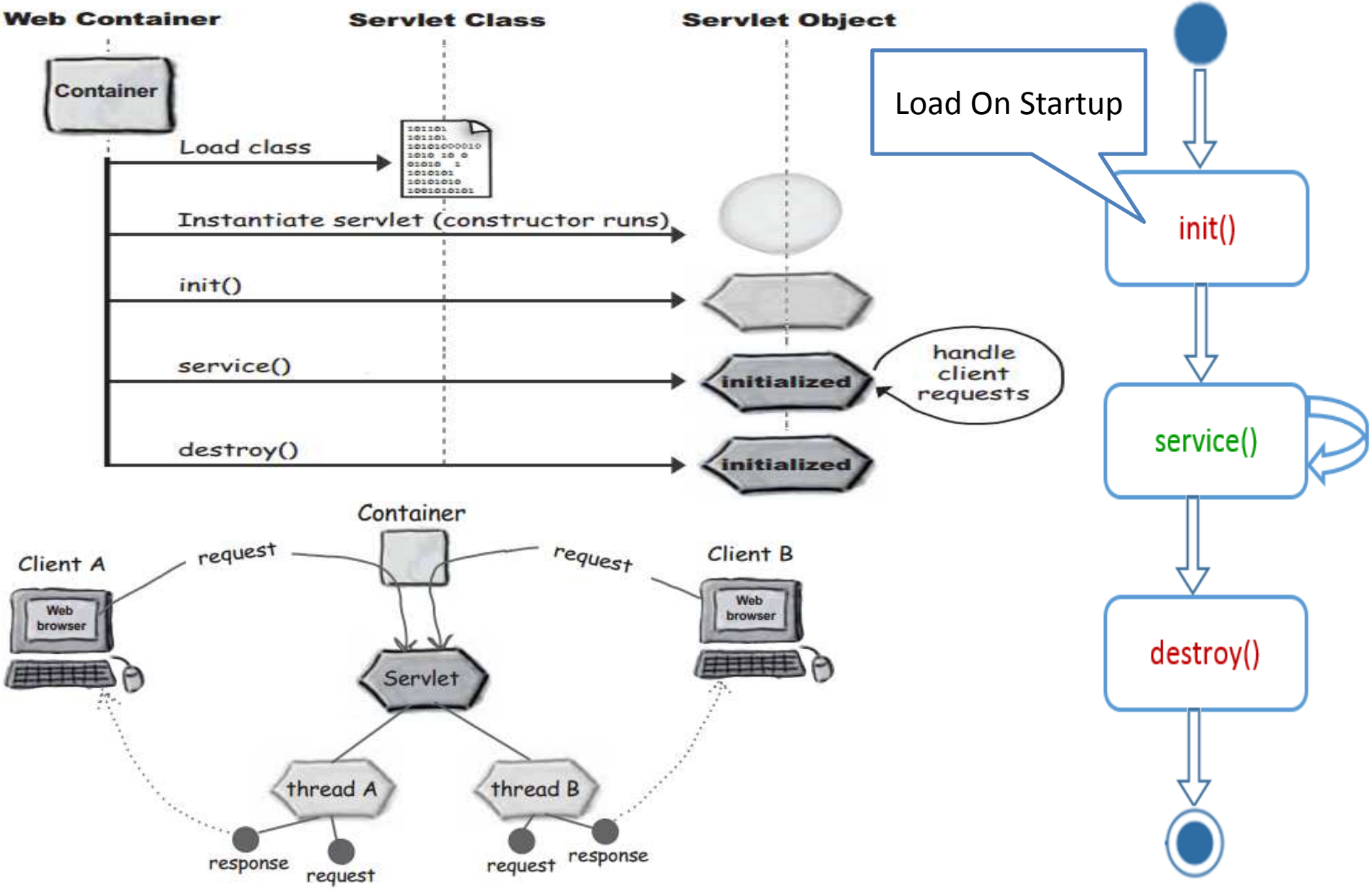
◉ Ciclo de vida



- ▷ interface *Servlet*
  - ✓ métodos que produzem os estados do ciclo de vida
  - ✓ estados
    - ⇒ inicializado: init()
    - ⇒ pronto: service()
    - ⇒ terminado: destroy()
- ▷ container *Servlet*
  - ✓ gerencia
    - ⇒ o ciclo de vida do servlet
    - ⇒ pool de recursos (thread pool da instância do servlet)
- ▷ Como funciona?
  - ⇒ verifica existência da instância
    - carrega a classe servlet, cria a instância do servlet, invoca init()
  - ⇒ chama o método service() para atender a requisição
  - ⇒ finaliza com o método destroy(), quando necessário



Servlets: Ciclo de vida



[06] CESPE - 2013 - TRT10

No que se refere ao desenvolvimento de aplicações HTML e JSP, julgue o próximo item.

No ciclo de vida de um *servlet*, o servidor recebe uma requisição e a repassa para o *container*, que a delega a um *servlet*. O *container* carrega a classe na memória, cria uma instância da classe do *servlet* e inicia a instância chamando o método init().

[07] - FCC - 2012 - TER-CE

No contexto do ciclo de vida de um servlet, considere:

I. Quando o servidor recebe uma requisição, ela é repassada para o *container* que, por sua vez, carrega a classe na memória e cria uma instância da classe do *servlet*.

II. Quando um *servlet* é carregado pela primeira vez para a máquina virtual *Java* do servidor, o método *init*() é invocado, para preparar recursos para a execução do serviço ou para estabelecer conexão com outros serviços.

[07] - FCC - 2012 - TER-CE

No contexto do ciclo de vida de um servlet, considere:

III. Estando o servlet pronto para atender as requisições dos clientes, o container cria um objeto de requisição (*ServletRequest*) e de resposta (*ServletResponse*) e depois chama o método *service*(), passando os objetos como parâmetros.

IV. O método *destroy*() permite liberar os recursos que foram utilizados, sendo invocado quando o servidor estiver concluindo sua atividade.

Está correto o que se afirma em:

a) I, II e III, apenas.
b) I, II e IV, apenas.
c) I, III e IV, apenas.
d) II, III e IV, apenas.
e) I, II, III e IV.

[08] FCC - 2013 - DPE SP

Um *Servlet Contêiner* controla o ciclo de vida de uma *servlet* onde são invocados três métodos essenciais: um para inicializar a instância da *servlet*, um para processar a requisição e outro para descarregar a *servlet* da memória. Os itens a seguir representam, nessa ordem, o que ocorre quando um usuário envia uma requisição HTTP ao servidor:

I. A requisição HTTP recebida pelo servidor é encaminhada ao *Servlet Contêiner* que mapeia esse pedido para uma *servlet* específica.

II. O *Servlet Contêiner* invoca o método *init* da *servlet*. Esse método é chamado em toda requisição do usuário à *servlet* não sendo possível passar parâmetros de inicialização.

[08] FCC - 2013 - DPE SP

III. O *Servlet Contêiner* invoca o método *service* da *servlet* para processar a requisição HTTP, passando os objetos *request* e *response*. O método *service* não é chamado a cada requisição, mas apenas uma vez, na primeira requisição do usuário à *servlet.*

IV. Para descarregar a *servlet* da memória, o *Servlet Contêiner* chama o método *unload*, que faz com que o *garbage collector* retire a instância da *servlet* da memória.

Está correto o que se afirma em

a) **I**, **II**, **III** e **IV**.
b) **I**, apenas.
c) **I** e **IV**, apenas.
d) **II**, **III** e **IV**, apenas.
e) **II** e **III**, apenas.

[09] CESPE - 2008 - MPE-RR

Considerando o código de uma servlet apresentado acima, julgue os itens a seguir, relativos a conceitos da linguagem e frameworks Java.

```
public class BookStoreServlet extends HttpServlet {
        public void service (HttpServletRequest request,
                        HttpServletResponse response)
                                throws ServletException, IOException
                                {
                RequestDispatcher dispatcher =
                                getServletContext().getRequestDispatcher("/bookstore/bookstore.html");
                if (dispatcher == null) {
                        System.out.println("There was no dispatcher");
                        response.sendError(response.SC_NO_CONTENT);
                } else {
                        System.out.println("There is a dispatcher");
                        HttpSession session = request.getSession();
                        dispatcher.forward(request, response);
                }
                                }
        public String getServletInfo() {
                return "The BookStore servlet returns the main web page " +
                                "for Duke's Bookstore.";
        }
}
```

[09] CESPE - 2008 - MPE-RR

Considerando o código de uma servlet apresentado acima, julgue os itens a seguir, relativos a conceitos da linguagem e frameworks Java.

Durante o funcionamento de uma aplicação *web* na qual esteja em uso a *servlet* acima declarada, cada pedido http enviado pelo *browser* e direcionado à *servlet* BookStoreServlet implicará a criação de uma nova instância da classe BookStoreServlet, bem como a criação de uma *thread* que invoca o método service(HttpServletRequest, HttpServletResponse), declarado no código apresentado.

◉ interface *javax.servlet.Servlet*

- ✓ deve ser implementada por qualquer servlet
- ✓ define os métodos do ciclo de vida
  - ▷ init()
    - ‣ `void init(ServletConfig config)`
      - – inicializa o servlet no container
  - ▷ service()
    - ‣ `void service(ServletRequest req, ServletResponse res)`
      - – responde as requests de clientes
  - ▷ destroy()
    - ‣ `void destroy()`
      - – finaliza servlet no container
- ✓ outros métodos
  - ▷ getServletConfig()
    - ‣ `ServletConfig getServletConfig()`
      - – retorna um ServletConfig com parâmetros de inicialização
  - ▷ getServletInfo()
    - ‣ `String getServletInfo()`
      - – retorna informações sobre servlet, tais como autor, versão, etc.

◉ interface *javax.servlet.ServletConfig*

- ✓ define um objeto usado para passar informações durante a inicialização
- ✓ fornece informações específicas para um servlet
  - Deployment Descriptor ou Annotations
- ✓ válido durante o ciclo de vida de seu servlet
  - ▷ `String getInitParameter(String name)`
    - – fornece o valor de um parâmetro de inicialização
  - ▷ `Enumeration<String> getInitParameterNames()`
    - – retorna os nomes dos parâmetros de inicialização do servlet
  - ▷ `ServletContext getServletContext()`
    - – retorna referência do ServletContext usado para interagir com o container
  - ▷ `String getServletName()`
    - – retorna o nome da instância corrente do servlet

◉ interface *javax.servlet.ServletContext*

- ✓ define um objeto usado para a comunicação com o container servlet
- ✓ há um objeto ServletContext por aplicação web
  - aplicação distribuída? um ServletContext por JVM
    - contexto não pode ser usado para compartilhar informações globais
- ✓ referência dentro de um objeto ServletConfig
  - `getServletContext()`

▷ parâmetros

- `String getInitParameter(String name)`
- `Enumeration<String> getInitParameterNames()`
- `boolean setInitParameter(String name, String value)`

▷ atributos

- `Object getAttribute(String name)`
- `Enumeration<String> getAttributeNames()`
- `void removeAttribute(String name)`
- `void setAttribute(String name, Object object)`

getAttribute(String name)	getInitParameter(String name)
getAttributeNames()	getInitParameterNames()
removeAttribute(String name)	setInitParameter(String name, String value)
setAttribute(String name, Object object)	

log(String msg)	getEffectiveMinorVersion()
log(String message, Throwable throwable)	getEffectiveSessionTrackingModes()
addFilter(String filterName, Class<? extends Filter> filterClass)	getFilterRegistration(String filterName)
addFilter(String filterName, Filter filter)	getFilterRegistrations()
addFilter(String filterName, String className)	getJspConfigDescriptor()
addListener(Class<? extends EventListener> listenerClass)	getMajorVersion()
addListener(String className)	getMimeType(String file)
addListener(<T extends EventListener> t)	getMinorVersion()
addServlet(String servletName, Class<? extends Servlet> servletClass)	getNamedDispatcher(String name)
addServlet(String servletName, Servlet servlet)	getRealPath(String path)
addServlet(String servletName, String className)	getRequestDispatcher(String path)
createFilter(Class<T> clazz)	getResource(String path)
createListener(Class<T> clazz)	getResourceAsStream(String path)
createServlet(Class<T> clazz)	getResourcePaths(String path)
declareRoles(String... roleNames)	getServerInfo()
getClassLoader()	getServletContextName()
getContext(String uripath)	getServletRegistration(String servletName)
getContextPath()	getServletRegistrations()
getDefaultSessionTrackingModes()	getSessionCookieConfig()
getEffectiveMajorVersion()	setSessionTrackingModes(Set<SessionTrackingMode> sessionTrackingModes)

getServlet(String name) Deprecated.	getServlets() Deprecated
getServletNames() Deprecated.	log(Exception exception, String msg) Deprecated.

◉ *javax.servlet.GenericServlet*

- ✓ classe abstrata que define um servlet independente de protocolo
- ✓ implementa métodos básicos para servlets
  - interfaces *Servlet* + *ServletConfig* e métodos *log()* de *ServletContext*
- ✓ fornece versões simples dos métodos do ciclo de vida *init()* e *destroy()*
  - service() é abstrato e precisa ser sobrescrito
- ▷ métodos
  - `void destroy()`
  - `void init() / init(ServletConfig config)`
  - `abstract void service(ServletRequest req, ServletResponse res)`
  - `ServletConfig getServletConfig()`
  - `String getServletInfo()`
  - `String getInitParameter(String name)`
  - `Enumeration<String> getInitParameterNames()`
  - `ServletContext getServletContext()`
  - `String getServletName()`
  - `void log(String msg) / log(String message, Throwable t)`

◉ *javax.servlet.http.HttpServlet*

- ✓ classe abstrata que define servlets específicos para o protocolo HTTP
- ✓ extende GenericServlet
  - ‣ sobrescreve `service(ServletRequest, ServletResponse)`
- ✓ implementa seu próprio método service()
  - ‣ `HttpServletRequest` e `HttpServletResponse`
  - ‣ plataforma desencoraja sobrescrever service()
- ▷ define métodos para lidar com requisições HTTP
  - ‣ `protected void doGet( )`
  - ‣ `protected void doPost( )`
  - ‣ `protected void doPut( )`
  - ‣ `protected void doDelete( )`
  - ‣ `protected void doHead( )`
  - ‣ `protected void doTrace( )`
  - ‣ `protected void doOptions( )`

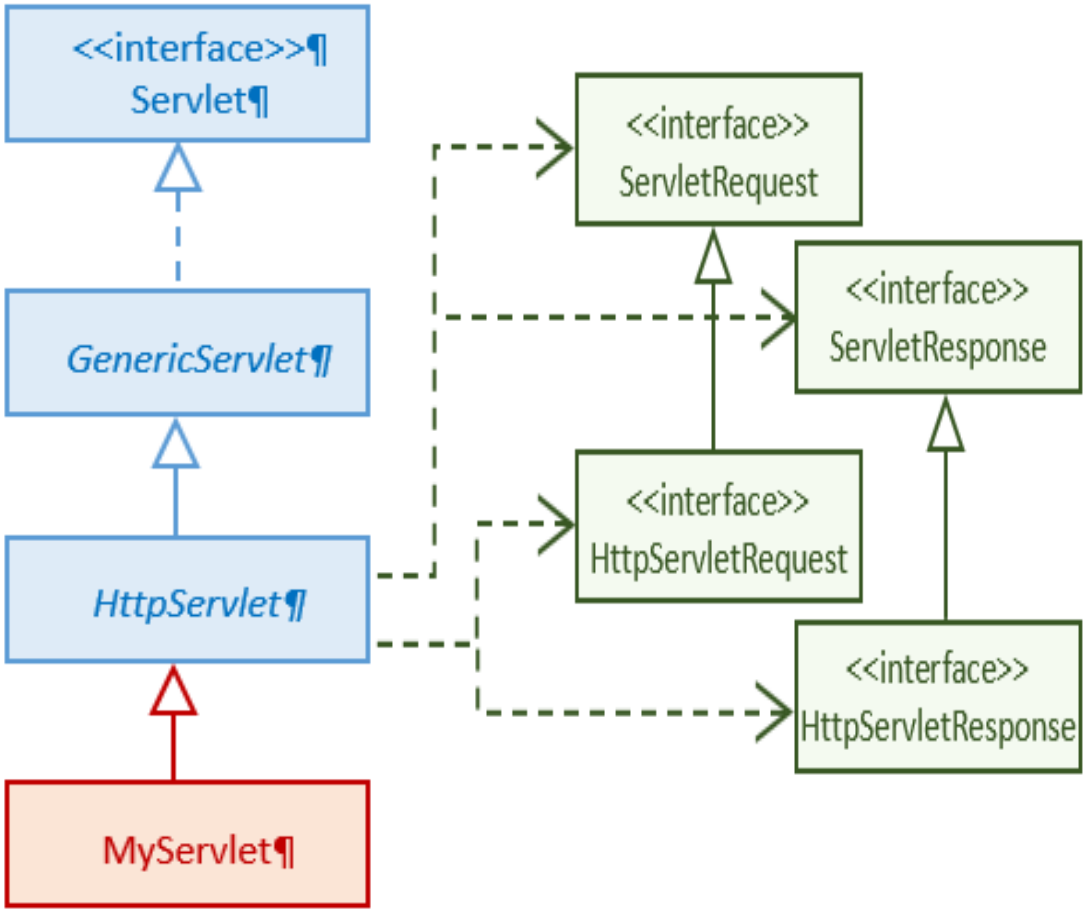
  - ‣ `protected long getLastModified(HttpServletRequest req)`

◉ *javax.servlet.http.HttpServlet*

- ✓ classe abstrata para HTTP servlets
  - ▷ servlets da aplicação estendem HttpServlet
    - ‣ `protected void doGet( )` ou `protected void doPost( )`

```
public class MyDoPostServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) {

    }

}
```

```
public class MyHttpServlet extends HttpServlet {

    protected void service(HttpServletRequest request,
            HttpServletResponse response){

    }

}
```



◉ *HttpServletRequest* e *HttpServletResponse*

- ✓ interfaces do pacote *javax.servlet.http*
- ✓ estendem
  - ‣ javax.servlet.ServletRequest
  - ‣ javax.servlet.ServletResponse
- ✓ instanciados pelo container
  - ‣ parâmetros de service(), doGet(), doPost, etc.

▷ HttpServletRequest

- ✓ recuperar dados da aplicação cliente

- String getParameter
- Object getAttribute
- String getHeader
- HttpSession getSession
- Enumeration<String> getParameterNames
- Enumeration<String> getAttributeNames
- Enumeration<String> getHeaders
- Cookie[] getCookies

◉ *HttpServletRequest* e *HttpServletResponse*

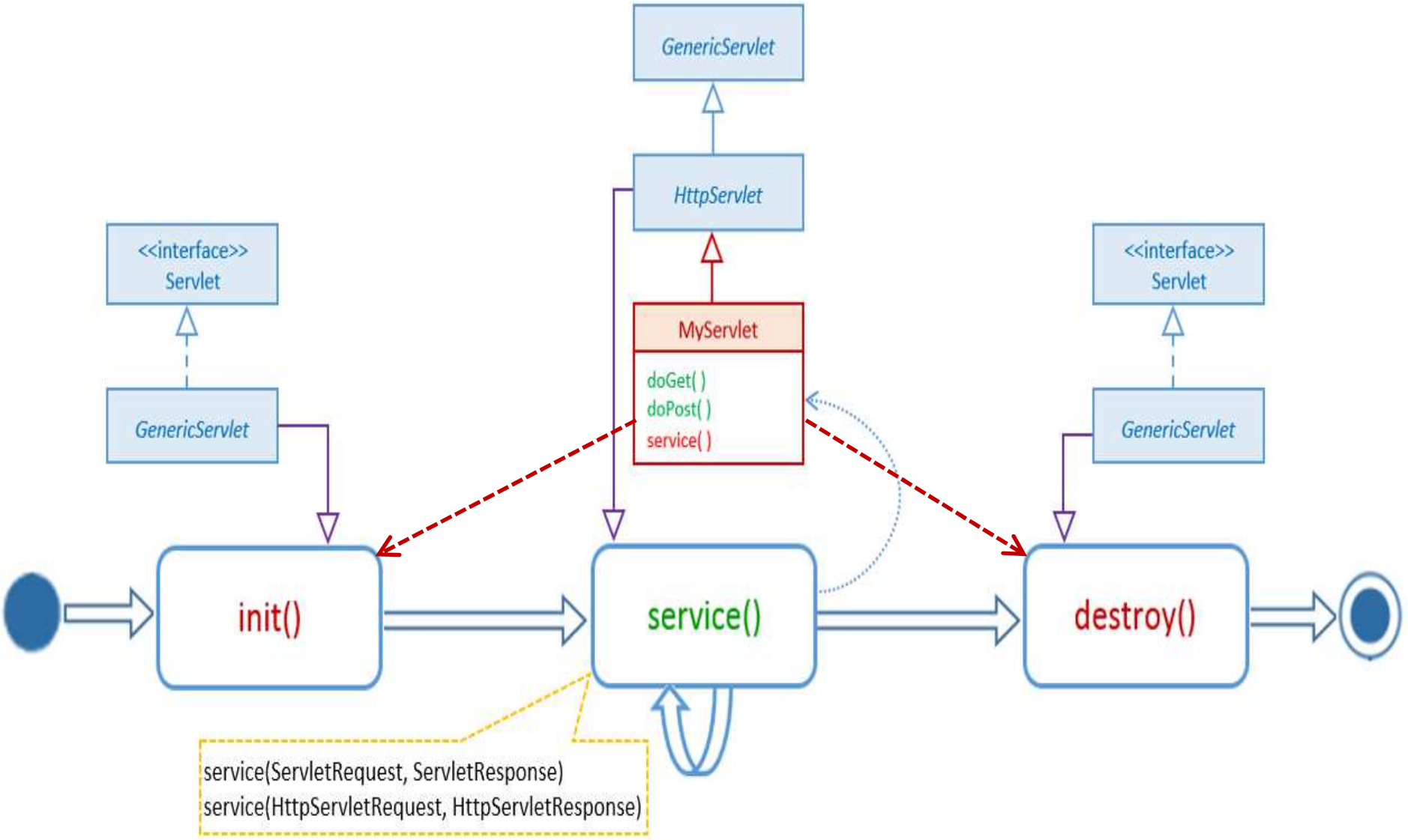
- ▷ HttpServletResponse
  - ✓ enviar dados para a aplicação cliente
  - ✓ comportamento padrão do método service()
    - ‣ extrair informações do request
    - ‣ acessar recursos externos
    - ‣ popular o response
      1. recuperar um fluxo de saída
      2. configurar os headers do response
      3. escrever a resposta no fluxo de saída
  - ▷ métodos da interface
    - ✓ fluxo de saída
      - ‣ `ServletOutputStream getOutputStream()` `java.io.PrintWriter getWriter()`
    - ✓ response headers
      - ‣ `addDateHeader(String name, long date)` `addHeader(String name, String value)`
      - ‣ `setDateHeader(String name, long date)` `setHeader(String name, String value)`
      - ‣ `setCharacterEncoding(String charset)` `setBufferSize(int size)`
      - ‣ `setContentType(String type)` `setContentLength(int len)`
      - ‣ `addCookie(Cookie cookie)` `setLocale(Locale loc)`

◉ *HttpServletRequest* e *HttpServletResponse*

▷ HTTPServletResponse

```
public class MyHttpServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,IOException{
        // recupera fluxo de saída
        PrintWriter out = response.getWriter();

        // configura headers response
        response.setIntHeader("Refresh", 5);
        response.setContentType("text/html");
        response.setBufferSize(8192);

        // escreve no fluxo de saída
        out.println("<html><head><title> MyDoPostServlet </title></head>");
        out.println("<body><h1> MyDoPostServlet - HttpServletResponse</h1>");
    }
}
```

Servlets: Ciclo de vida (*full*)



[10] VUNESP - 2013 - FUNDUNESP

Considere o *Servlet* a seguir:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ClasseServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,HttpServletResponse response){
            response.write("<html>");
            response.write("<body>");
            response.write("Servlet em operação!");
            response.write("</body>");
            response.write("</html>");
    }
}
```

Sobre o código do *Servlet*, é possível afirmar que:

[10] VUNESP - 2013 - FUNDUNESP

Sobre o código do *Servlet*, é possível afirmar que:

a) ao ser executado por um contêiner de *Servlet*, será exibida uma tela em branco no navegador.

b) ao ser executado por um contêiner de *Servlet*, será exibida a mensagem “Servlet em operação!” na tela do navegador.

c) não pode ser compilado, pois a classe HttpServletResponse não possui o método write.

d) não pode ser compilado, pois HttpServlet é uma interface e, portanto, não pode ser estendida por uma classe.

e) o conteúdo exibido na tela do navegador não será codificado corretamente, pois a codificação da página não foi informada.

[11] FCC - 2006 - BACEN [adaptada]

Para ser um servlet, uma classe deve estender a classe ..I.. e sobrescrever o método doGet ou doPost (ou ambos), dependendo se os dados estão sendo enviados por uma ação GET ou por uma ação POST. Estes métodos tomam dois argumentos: um ..II.. e um ..III.. em sua execução.

Preenchem correta e respectivamente as lacunas I, II e III:

I	II	III
a) HttpJspServlet	XmlHttpServlet	XmlHttpServletResponse
b) JspServlet	HttpServletResponse	HttpServletRequest
c) HttpServletRequest	XmlHttpServletRquest	XmlHttpServletResponse
d) HttpServlet	HttpServletRequest	HttpServletResponse
e) HttpServlet	HttpServletRequest	HttpServletResponse.write

[12] FGV - 2013 - ALEMA

Considere que o código *Servlet* a seguir será compilado e instalado em um servidor JEE.

```
import java.io.*;
import javax.servlet.http.*;

public class Teste extends HttpServlet {

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
                    throws IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<h1>Ola!!</h1>");
        out.close();
    }
}
```

[12] FGV - 2013 - ALEMA

Considere que o código *Servlet* a seguir será compilado e instalado em um servidor JEE.

A partir das informações acima, assinale a afirmativa correta.

a) A compilação falha porque o método *doGet* é protegido.

b) O código compila e instala sem problemas.

c) A compilação falha porque o pacote que define *HttpServlet* não está incluído na seção *import*.

d) A instalação falha porque a classe *Teste* não contém o método obrigatório *destroy*.

e) A compilação falha porque o método *doGet* não declara uma exceção do tipo *ServletException*.

[12] FGV - 2013 - ALEMA

```
38      protected void doGet(HttpServletRequest request,
39              HttpServletResponse response) {
40
41          response.setContentType("text/html; charset=UTF-8");
42          PrintWriter out = response.getWriter();
43          request.getRequestDispatcher("Teste")
44          response.sendRedirect("teste");
45          out.println("<h1>Ola!!</h1>");
46          out.close();
47      }
48  }
```

Unhandled exception type IOException

2 quick fixes available:

- Add throws declaration
- Surround with try/catch

Press 'F2' for focus

```
7   protected void doGet(HttpServletRequest request,
8           HttpServletResponse response)
9               throws IOException, EOFException {
10      response.setContentType("text/html; charset=UTF-8");
11      PrintWriter out = response.getWriter();
12      out.println("<h1>Ola!!</h1>");
13      out.close();
14  }
```

[12] FGV - 2013 - ALEMA

```
38    protected void doGet(HttpServletRequest request,
39            HttpServletResponse response)
40                    throws IOException,SQLException {
41
42        response.setContentType("text/h
43        PrintWriter out = response.getW
44        request.getRequestDispatcher("T
45        response.sendRedirect("teste");
46        out.println("<h1>Ola!!</h1>");
47        out.close();
48    }
49 }
```

Exception SQLException is not compatible with throws clause in HttpServlet.doGet(HttpServletRequest, HttpServletResponse)

1 quick fix available:

Remove exceptions from 'doGet(..)'

Press 'F2' for focus

```
38    protected void doGet(HttpServletRequest request,
39            HttpServletResponse response)
40                    throws IOException, Exception {
41
42        response.setContentType("text/ht
43        PrintWriter out = response.getWr
44        out.println("<h1>Ola!!</h1>");
45        out.close();
46    }
47 }
```

Exception Exception is not compatible with throws clause in HttpServlet.doGet(HttpServletRequest, HttpServletResponse)

1 quick fix available:

Remove exceptions from 'doGet(..)'

Press 'F2' for focus

[13] FCC - 2013 - ALERN

Servlets são componentes da plataforma Java EE que recebem no servidor requisições dos computadores cliente. Considere uma aplicação *web* composta por uma página HTML e uma *servlet*. A página contém no seu corpo o seguinte formulário:

```
<form method="post" action="Controle">
    <label>
        ID: <input type="text" name="id" />
    </label>
    <input type="submit" value="Enviar" />
</form>
```

Ao clicar no botão **Enviar**, o conteúdo do campo é submetido à *servlet* **Controle.java** no servidor. Nessa *servlet,* há um objeto **request** da interface **HttpServletRequest.**

Para receber o conteúdo do campo texto do formulário e armazenar em uma variável, pode-se utilizar a instrução

a) int id = Integer.parseInt(request.getParameter("id"));

b) int id = request.getParameter("id");

c) int id = request.getInt("id");

d) String id = request.getParameter("ID");

e) String id = request.getString("id");

[14] FCC - 2013 - DPE SP

Considere uma aplicação *web* desenvolvida utilizando-se o *Java EE 6* que contém dois arquivos, uma página de abertura de um site (chamada index.html) e uma classe *servlet* (Controle.java):

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Teste</title>
    </head>
    <body>
        <form method ="post" action="Controle">
            <p>Interesses: <br/>
            <label><input type="checkbox" value="Livros" name="interesses"/>Livros</label>
            <label><input type="checkbox" value="Revistas" name="interesses"/>Revistas</label>
            <label><input type="checkbox" value="Teatro" name="interesses"/>Teatro</label>
            </p>
            <p> <input type="submit" value="Enviar"/></p>
        </form>
    </body>
</html>
```

[14] FCC - 2013 - DPE SP

Considere uma aplicação *web* desenvolvida utilizando-se o *Java EE 6* que contém dois arquivos, uma página de abertura de um site (chamada index.html) e uma classe *servlet* (Controle.java):

Controle.java

```
@WebServlet(name = "Controle", urlPatterns = {"/Controle"})
public class Controle extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
}
```

[14] FCC - 2013 - DPE SP

Considere uma aplicação *web* desenvolvida utilizando-se o *Java EE 6* que contém dois arquivos, uma página de abertura de um site (chamada index.html) e uma classe *servlet* (Controle.java):

Com base nessa aplicação e na plataforma *Java EE 6* é correto afirmar que

a) a instrução para receber no método *processRequest* da *servlet* os dados selecionados no formulário é String[3] interesses = request.getParameter("interesses");.

b) ao submeter os dados selecionados no formulário HTML, esses dados serão recebidos no método doGet da *servlet,* pois esse é o método padrão para requisições HTTP em uma aplicação *web*.

[14] FCC - 2013 - DPE SP

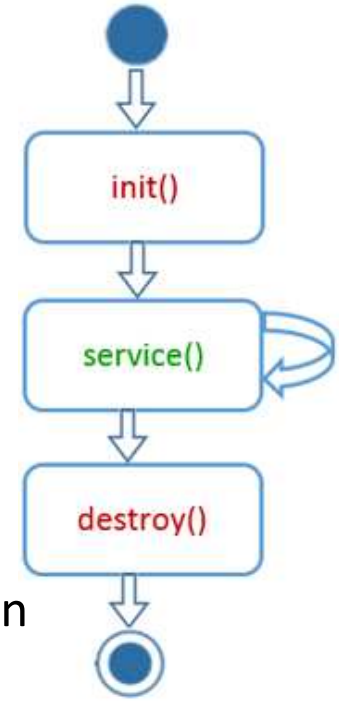
Com base nessa aplicação e na plataforma *Java EE 6* é correto afirmar que

c) os pacotes javax.servlet e javax.servlet.http oferecem interfaces e classes para escrever *servlets*. A classe javax.servlet.http.HttpServlet fornece métodos, tais como o doGet e o doPost que foram sobrescritos na *servlet* Controle.java.

d) o código que deve ser utilizado no método processRequest da *servlet* para receber e exibir os dados selecionados no formulário é String[] interesses = request.getParameterValues ("interesses"); for (int i=0; i <= interesses.size(); i++) { out.println(interesses[i]); }.

e) os métodos doPost e doGet devem ser excluídos, pois os dados recebidos por esses métodos no objeto *request* são passados para o método processRequest, logo, basta o método processRequest para receber os dados das requisições.

Servlets: Ciclo de vida



- Eventos
  - ✓ plataforma permite monitorar ciclo de vida
- ▷ Objetos listener
  - ✓ métodos invocados em eventos do ciclo de vida
  - ✓ como funciona?
    - ⇒ desenvolvedor:
      - ‣ implementa o listener e o registra no DD ou com annotation
    - ⇒ container:
      - ‣ instancia classe de evento e passa para o listener

Evento	Interface Listener	Class do Evento	Escopo
inicialização e destruição	ServletContextListener	ServletContextEvent	web context
atributo adicionado, removido ou substituído	ServletContextAttributeListener	ServletContextAttributeEvent	web context
criação, invalidação, ativação, passivação, e tempo de espera	HttpSessionListener, HttpSessionActivationListener	HttpSessionEvent	session
atributo adicionado, removido ou substituído	HttpSessionAttributeListener	HttpSessionBindingEvent	session
request servlet processada por componentes web	ServletRequestListener	ServletRequestEvent	request
atributo adicionado, removido ou substituído	ServletRequestAttributeListener	ServletRequestAttributeEvent	request

Servlets: Ciclo de vida

- Eventos

```
12 @WebListener
13 public class MyListener implements ServletContextListener {
14
15     public void contextInitialized(ServletContextEvent event) {
16         ServletContext context = event.getServletContext();
17
18         String url = context.getInitParameter("url");
19         String user_name = context.getInitParameter("user_name");
20         String password = context.getInitParameter("password");
21
22         DBConnectionManager dbCon = new DBConnectionManager(url, user_name, password);
23
24         context.setAttribute("myDBCon", dbCon);
25     }
26
27     public void contextDestroyed(ServletContextEvent event) {
28         ServletContext context = event.getServletContext();
29
30         DBConnectionManager dbCon = (DBConnectionManager) context.getAttribute("myDBCon");
31         dbCon.closeConnection();
32     }
33
34 }
```

Servlets: Ciclo de vida

- Eventos

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <context-param>
        <param-name>DBuserName</param-name>
        <param-value>leomarc</param-value>
    </context-param>
    <context-param>
        <param-name>DBpassword</param-name>
        <param-value>P@55w0rD</param-value>
    </context-param>
    <context-param>
        <param-name>DBurl</param-name>
        <param-value>jdbc:oracle:thin:@//localhost:1521/XE</param-value>
    </context-param>
    <listener>
        <listener-class>MyListener</listener-class>
    </listener>
</web-app>
```

Servlets: Escopo de Objetos

- ✓ indica
  - tempo de vida do objeto
  - recurso que pode acessar o objeto
- ✓ objetos
  - armazenados em um nível de escopo
  - atributos do escopo

- ▷ níveis de escopo
  - ⇒ Web context (`ServletContext`)
    - acessado por todos componentes da aplicação (application)
  - ⇒ Session (`HttpSession`)
    - acessado em uma sessão específica
    - múltiplas requests
  - ⇒ Request (`ServletRequest`)
    - acessado por uma request específica
    - múltiplas páginas
  - ⇒ Page (`JspContext`)
    - acessado por uma página específica

[15] FCC - 2008 - TCE-SP

As conexões ao banco de dados são estabelecidas no ciclo de vida de um Servlet na fase de

a) preparação.

b) inicialização.

c) finalização.

d) carga de arquivos.

e) atendimento às requisições.

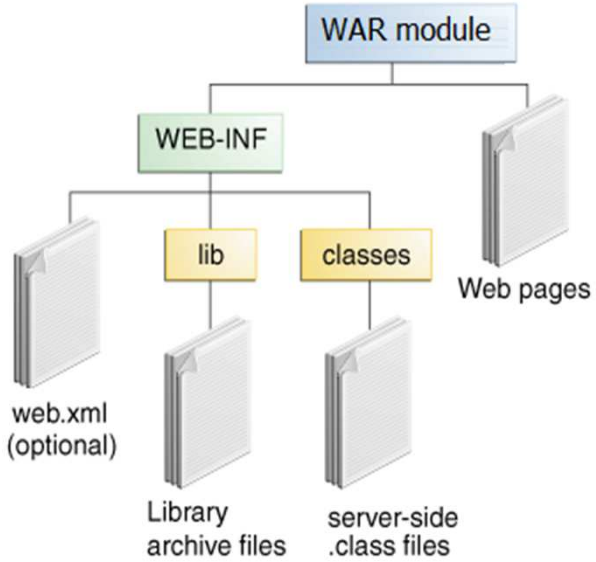
[16] CESPE - 2011 - Correios

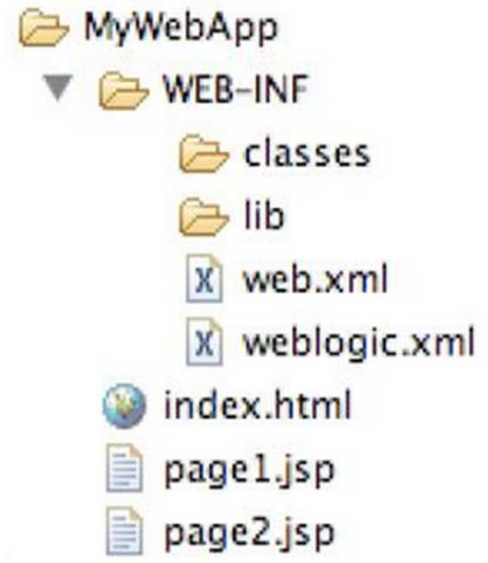
Acerca dos fundamentos, características e topologias típicas em ambientes com alta disponibilidade e escalabilidade e da arquitetura J2EE, julgue os próximos itens.

Entre outras aplicações, os *servlets* são utilizados para escrever aplicativos *web* J2EE dinâmicos em servidores *web*. Um *servlet* pode utilizar seus recursos para realizar ações como, por exemplo, usar os registros (*logging*) para permitir que o servidor possa autenticar usuários.

◉ Deployment Descriptor

- ▷ descritor de implantação (opcional)
  - ✓ XML com as configurações dos componentes
  - ✓ informações declarativas
    - ☑ alteração sem recompilar o servlet
  - ✓ lidos em tempo de execução pelo container
- ▷ tipos
  - ✓ descritor JEE e descritor de runtime
- ▷ Módulo Web (Web ARchive)
  - ✓ WEB-INF
    - ⇒ web.xml (JEE) e xxxx.xml (runtime)
- ▷ web.xml (JEE) (opcional?)
  - ✓ obrigatório
    - ‣ JEE5: servlets, filters ou listeners
    - ‣ JEE6: JSF, segurança, sobrescrever annotations
  - ✓ tem precedência sobre annotations e web-fragment





◉ web.XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app metadata-complete="true" version="3.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID">

    <display-name>Example App</display-name>

    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>

    <context-param>
        <param-name>DBuserName</param-name>
        <param-value>leomarc</param-value>
    </context-param>
    <context-param>
        <param-name>DBpassword</param-name>
        <param-value>P@55w0rD</param-value>
    </context-param>

    <listener>
        <listener-class>MtContextListener</listener-class>
    </listener>
```

web.XML

```
26      <servlet>
27          <display-name>Servlet1</display-name>
28          <servlet-name>Servlet1</servlet-name>
29          <servlet-class>test.Servlet1</servlet-class>
30          <init-param>
31              <param-name>sleep-time-in-seconds</param-name>
32              <param-value>10</param-value>
33          </init-param>
34          <load-on-startup>1</load-on-startup>
35      </servlet>
36
37      <servlet-mapping>
38          <servlet-name>Servlet1</servlet-name>
39          <url-pattern>/Servlet1</url-pattern>
40      </servlet-mapping>
41
42      <error-page>
43          <error-code>404</error-code>
44          <location>/error404.jsp</location>
45      </error-page>
46
47      <welcome-file-list>
48          <welcome-file>index.html</welcome-file>
49          <welcome-file>index.htm</welcome-file>
50          <welcome-file>index.jsp</welcome-file>
51          <welcome-file>default.html</welcome-file>
52          <welcome-file>default.htm</welcome-file>
53          <welcome-file>default.jsp</welcome-file>
54          <welcome-file>index.swf</welcome-file>
55          <welcome-file>splash.jsp</welcome-file>
56      </welcome-file-list>
57
```

web.XML

```
58      <security-role>
59          <role-name>admin</role-name>
60      </security-role>
61
62      <security-constraint>
63          <web-resource-collection>
64              <web-resource-name>Protected Area</web-resource-name>
65              <url-pattern>/private/*</url-pattern>
66          </web-resource-collection>
67
68          <auth-constraint>
69              <role-name>admin</role-name>
70          </auth-constraint>
71      </security-constraint>
72
73      <security-constraint>
74          <web-resource-collection>
75              <web-resource-name>Entire Application</web-resource-name>
76              <url-pattern>/*</url-pattern>
77          </web-resource-collection>
78
79          <user-data-constraint>
80              <transport-guarantee>CONFIDENTIAL</transport-guarantee>
81          </user-data-constraint>
82      </security-constraint>
83
84  </web-app>
```

◉ web-fragment.XML

- ▷ Web Module Deployment Descriptor Fragments
  - ✓ Servlet 3.0
    - ☑ antes: todo metadado da aplicação ficava no web.xml
  - ✓ permite dividir
    - ⇒ configurações de frameworks e bibliotecas (web-fragment.xml)
    - ⇒ configurações da aplicação
  - ✓ facilita manutenção do web.XML
    - ☑ melhor "*pluggability*" e menos configurações para desenvolvedores
- ▷ deploy
  - ⇒ diretório META-INF do arquivo JAR (WEB-INF/lib)
- ▷ elementos
  - ⇒ raiz: <web-fragment>
  - ⇒ sub-elementos = web.xml
    - ☑ <ordering> web.xml: <absolute-ordering>
- ▷ metadata-complete
  - ⇒ TRUE: annotations do JAR não serão processadas

◉ Servlet: @nnotations

✓ modificador inserido na classe java (metadados)

✓ facilita o deploy diminuindo configurações em XML

☑ especificação: classes WEB-INF/classes e JAR em WEB-INF/lib

▷ estrutura

✓ pacote: *javax.servlet.annotation*

`@WebServlet`

- deve estender `javax.servlet.http.HttpServlet`
- requer atributo `value` ou `urlPatterns`

- `String name`
- `String[] urlPatterns`
- `String[] value`
- `String displayName`
- `String description`
- `boolean asyncSupported`
- `WebInitParam[] initParams`
- `int loadOnStartup`

```
18 @WebServlet(
19         name = "MyDoPostServlet",
20         description = "This is annotated servlet",
21         urlPatterns = {"/doPostServlet","/sendFile", "/uploadFile"}
22         )
```

Servlet: @nnotations

@WebFilter

- deve implementar `javax.servlet.Filter`
- requer atributo `value`, `urlPatterns` ou `servletNames`
- só tem url? `value` é recomendado

```
@WebFilter("/uploadFile")
@WebServlet("/doPostServlet")
```

- `String filterName`
- `String[] urlPatterns`
- `String[] value`
- `String displayName`
- `String description`
- `String[] servletNames`
- `boolean asyncSupported`
- `DispatcherType[] dispatcherTypes`
- `WebInitParam[] initParams`

@WebInitParam

- parâmetros passados para Servlets ou Filters

```
21 @WebFilter(
22         filterName = "AdminFilter",
23         description = "Filter all admin URLs",
24         servletNames = {"MyOwnServlet", "UploadServlet"},
25         urlPatterns = {"/admin/*", "/uploadFile"},
26         dispatcherTypes = {DispatcherType.REQUEST, DispatcherType.FORWARD},
27         initParams ={
28                 @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
29                 @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
30                 }
31         )
```

Servlet: @nnotations

@WebListener

- capturar eventos no ciclo de vida do Servlet
- deve implementar uma destas interfaces
  - javax.servlet.ServletContextListener
  - javax.servlet.ServletContextAttributeListener
  - javax.servlet.ServletRequestListener
  - javax.servlet.ServletRequestAttributeListener
  - javax.servlet.http.HttpSessionListener
  - javax.servlet.http.HttpSessionAttributeListener
- atributo: `String value` (opcional)

@MultipartConfig

- indica que o request do Servlet espera um tipo `multipart/form-data`
- usado para upload de arquivos no servlet 3.0
- HttpServletRequest
  - `Part getPart(name)`
  - `Collection<Part> getParts()`
- atributos
  - `int fileSizeThreshold`
  - `String location`
  - `long maxFileSize`
  - `long maxRequestSize`

Servlet: @nnotations

@MultipartConfig

```
11 @WebServlet(name = "FileUpload", urlPatterns = { "/FileUpload" })
12 @MultipartConfig(
13         location = "C:\\tmp",
14         fileSizeThreshold = 1024 * 1024,
15         maxFileSize = 1024 * 1024 * 5,
16         maxRequestSize = 1024 * 1024 * 5 * 5
17         )
18
19 public class FileUpload extends HttpServlet {
20     public void service(HttpServletRequest req, HttpServletResponse resp)
21             throws ServletException, IOException {
22         String filename=req.getParameter("filename");
23
24         req.getPart("content").write("C:/data/"+filename+".txt");
25
26         if(req.getPart("content")!=null){
27             PrintWriter out = resp.getWriter();
28             out.write("<h3>File uploaded successfully</h3>");
29         }
30     }
31 }
```

```
<FORM action="FileUpload" enctype="multipart/form-data" method="POST">
        File Name:    <INPUT type="text" name="filename"><br/>
        Upload File:  <INPUT type="file" name="content"><br/>
        <INPUT type="submit" value="Submit">
</FORM>
```

[17] FCC - 2012 - TJ-PE

Sobre a plataforma Java EE 6, é correto afirmar:

a) Simplifica a implantação sem a necessidade de descritores de implantação, com exceção do descritor de implantação exigido pela especificação servlet, o arquivo web.xml.

b) Necessita do descritor de implantação ejb-jar.xml e entradas relacionadas aos *webservices* no arquivo web.xml.

c) Faz uso de anotações (*annotations*). Anotações são modificadores Java, semelhantes aos públicos e privados, que devem ser especificados nos arquivos de configuração XML.

d) A especificação EJB 3, que é um subconjunto da especificação Java EE, define anotações apenas para o tipo *bean*.

e) Anotações são marcados com um caracter # (cerquilha).

[18] CESGRANRIO - 2012 - CEF

Qual é o objetivo da anotação @WebServlet, presente no JEE v6?

a) Registrar um filtro.

b) Registrar um Context Listener.

c) Definir parâmetros de inicialização para Servlets e filtros.

d) Substituir os mapeamentos de Servlet presentes no web.xml.

e) Documentar uma Servlet gerando uma descrição do que a mesma realiza.

◉ Servlet: Filtros

- ✓ permite modificar os objetos request e response
  - ☑ request antes de ser entregue ao servlet
  - ☑ response depois do servlet atuar
- ✓ tem uso diferente de um servlet
  - ☑ não é usado para responder requisições
  - ☑ modificam/adaptam request e response
- ✓ baixo acoplamento
  - ☑ não deve depender de um determinado recurso
  - ☑ pode ser ligado a qualquer tipo de recurso web
  - ☑ desacopla RNFs da lógica do negócio
    - ‣ auditoria, segurança, etc
- ✓ coesão
  - ☑ um filtro encapsula apenas uma responsabilidade
  - ☑ são autocontidos e transparentes
  - ☑ permite o encadeamento de filtros

Servlet: Filtros

- estrutura
  - ✓ pacote: *javax.servlet*

FilterConfig

- String getFilterName()
- Enumeration<String> getInitParameterNames()
- String getInitParameter(String name)
- ServletContext getServletContext()

FilterChain

- doFilter(ServletRequest, ServletResponse)

Filter

- init(FilterConfig)
- destroy()

- **doFilter**(ServletRequest, ServletResponse, FilterChain)
  - implementa a funcionalidade do filtro
  - chamado pelo container para aplicar o filtro

◉ Servlet: Filtros

◉ doFilter()

- rotina
  1. examina headers do request
  2. modifica objetos request e response
  3. invoca a próxima entidade

     não invoca outra entidade
  4. examina headers do response

☑ chain.doFilter(request,response)

- código antes executa antes do servlet (ida)
- código depois executa depois do servlet (volta)

☑ Filter.doFilter X FilterChain.doFilter

- Filter: método invocado pelo container

  doFilter(ServletRequest,ServletResponse,FilterChain)
- FilterChain: chama outra entidade da pilha
  - invocado no método doFilter dentro da classe Filtro

    doFilter(ServletRequest,ServletResponse)

◉ Servlet: Filtros

```
public class AuthenticationFilter implements Filter {
  private ServletContext context;

  public void init(FilterConfig fConfig) throws ServletException {
    this.context = fConfig.getServletContext();
    this.context.log("AuthenticationFilter inicializado");
  }

  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;

    String uri = req.getRequestURI();
    this.context.log("Requested Resource::"+uri);

    HttpSession session = req.getSession(false);

    if(session == null && !(uri.endsWith("html") || uri.endsWith("LoginServlet"))){
      this.context.log("Unauthorized access request");
      res.sendRedirect("login.html");
    }else{
      chain.doFilter(request, response);
    }
  }

  public void destroy() {
    this.context.log("AuthenticationFilter finalizado");
  }
}
```

Servlet: Filtros

Deployment Descriptor

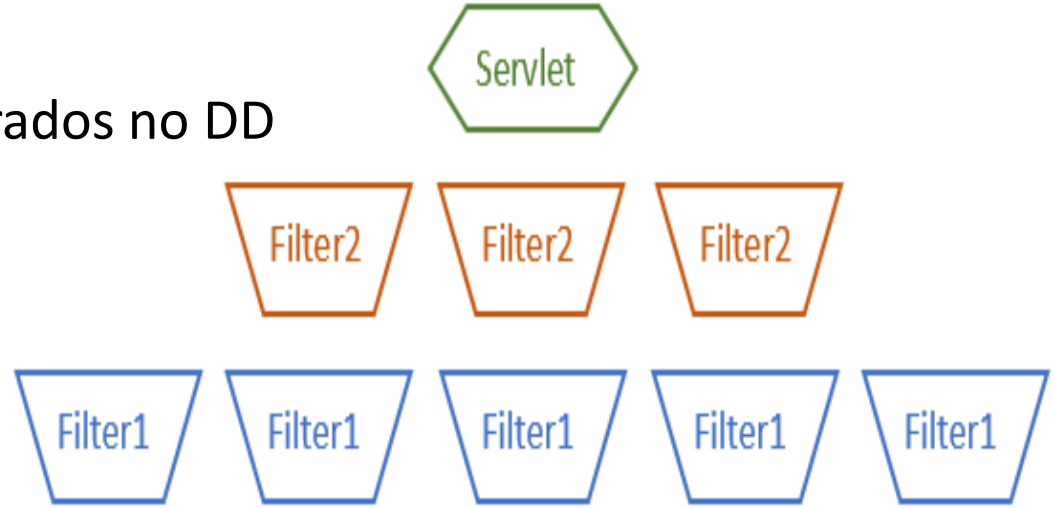
- <filter>
  - .<filter-name>
  - .<filter-class>
  - .<init-param>
- <filter-mapping>
  - .<filter-name>
  - .<url-pattern>
  - .<servlet-name>
  - .<dispatcher>
    - **REQUEST**
    - INCLUDE
    - FORWARD
    - ERROR
    - ASYNC

@WebFilter

- .filterName
- .servletNames
- .dispatcherTypes
- .urlPatterns ou value

Encadeamento

- ✓ na ordem que são declarados no DD
  - 1º url-patterns
  - 2º servlet-name



@WebFilter não ordena

◉ Servlet: Filtros

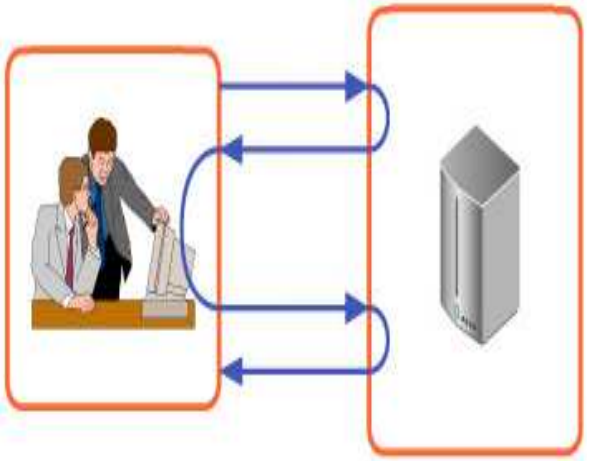
```
@WebFilter(
        filterName = "AdminFilter",
        description = "Filter all admin URLs",
        servletNames = {"MyOwnServlet", "UploadServlet"},
        urlPatterns = {"/admin/*", "/uploadFile"},
        dispatcherTypes = {DispatcherType.REQUEST, DispatcherType.FORWARD},
        initParams ={
                @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
                @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
                }
        )
```

```
<filter>
    <filter-name>AdminFilter</filter-name>
    <filter-class>MyFilterExample</filter-class>
    <init-param>
        <param-name>saveDir</param-name>
        <param-value>D:/FileUpload</param-value>
    </init-param>
    <init-param>
        <param-name>allowedTypes</param-name>
        <param-value>jpg,jpeg,gif,png</param-value>
    </init-param>
</filter>
```

```
<filter-mapping>
    <filter-name>AdminFilter</filter-name>
    <servlet-name>MyOwnServlet</servlet-name>
    <servlet-name>UploadServlet</servlet-name>
    <url-pattern>/admin/*</url-pattern>
    <url-pattern>/uploadFile</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

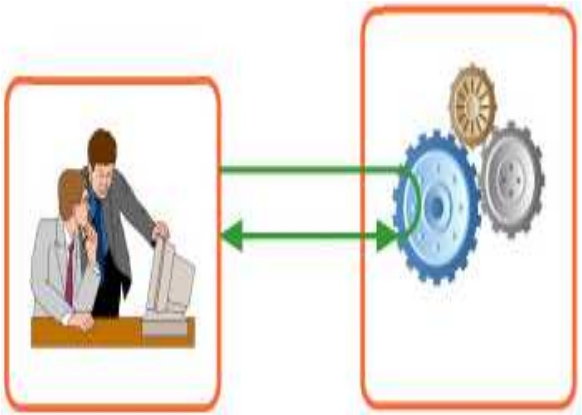
Servlet: Invocação de recursos

- ✓ permite delegar processamento da requisição
- ✓ invoca recursos direta ou indiretamente



SendRedirect

- ✓ invocação indireta ⇒ lado cliente
- ✓ altera URL do browser ⇒ não transparente
- ✓ método de HttpServletResponse
  - `sendRedirect(String)`
- ✓ deve ser invocado antes de escrever na saída (outputstream)



RequestDispatcher

- ✓ invocação direta ⇒ lado servidor
- ✓ não altera URL do browser ⇒ transparente
- ✓ interface

Servlet: Invocação de recursos

- RequestDispatcher
  - ✓ interface de javax.servlet
    - `forward(ServletRequest,ServletResponse)`
      - ✓ transferência permanente do controle para outro recurso
      - ✓ deve ser invocado antes de escrever na saída (outputstream)
    - `include(ServletRequest,ServletResponse)`
      - ✓ inclui conteúdo gerado por outro recurso na resposta
      - ✓ transferência temporária do controle para outro recurso
  - ✓ Como obter objeto RequestDispatcher?
    - 1º ServletContext
      - `getRequestDispatcher(String path)`
      - `getNamedDispatcher(String ServletName)`
    - 2º ServletRequest
      - `getRequestDispatcher(String path)`

```
ServletRequest.getRequestDispatcher("header.html")
ServletContext.getRequestDispatcher("/garden/header.html")
```

◉ Servlet: Invocação de recursos

```
21  protected void doGet(HttpServletRequest request, HttpServletResponse response)
22          throws ServletException, IOException  {
23
24      PrintWriter out = response.getWriter();
25      out.println("Início do Redirecionamento de fluxo<br/>");
26      out.flush();
27
28      String action = request.getParameter("action");
29      if (action != null) {
30          RequestDispatcher rd = request.getRequestDispatcher("MyServlet");
31          rd.include(request, response);
32      }
33      out.println("Término do Redirecionamento de fluxo<br/>");
34      out.close();
35  }
```

```
21  protected void doGet(HttpServletRequest request, HttpServletResponse response)
22          throws ServletException, IOException  {
23
24      PrintWriter out = response.getWriter();
25      out.println("Início do Redirecionamento de fluxo<br/>");
26      out.flush();
27
28      String action = request.getParameter("action");
29      if (action != null) {
30          RequestDispatcher rd = request.getRequestDispatcher("MyServlet");
31          rd.forward(request, response);
32      }
33      out.println("Término do Redirecionamento de fluxo<br/>");
34      out.close();
35  }
```

IllegalStateException

Servlet: Invocação de recursos

```
21    protected void doGet(HttpServletRequest request, HttpServletResponse response)
22            throws ServletException, IOException  {
23
24        String action = request.getParameter("action");
25
26        if (action != null) {
27            RequestDispatcher rd = request.getRequestDispatcher("MyServlet");
28            rd.forward(request, response);
29        }
30    }
```

```
21  protected void doGet(HttpServletRequest request, HttpServletResponse response)
22          throws ServletException, IOException  {
23
24      String action = request.getParameter("action");
25
26      if (action != null) {
27          response.sendRedirect("MyServlet?user=foo");
28      }
29  }
```

Servlet: Sessões

HttpSession

- ✓ interface de javax.servlet.http
- ✓ métodos para identificar requests de um cliente

```
•getAttribute(String)      •getLastAccessedTime()      •isNew()
•getAttributeNames()       •getMaxInactiveInterval()   •removeAttribute(String)
•getCreationTime()         •getServletContext()        •setAttribute(String, Object)
•getId()                   •invalidate()               •setMaxInactiveInterval(int)
```

Deprecated

```
•getSessionContext()       •getValue(String)           •getValueNames()
•putValue(String, Object)  •removeValue(String)
```

HttpServletRequest.getSession()

- ✓ retorna a sessão atual ou cria uma nova
- ✓ container controla o sessionID
  - ☑ objeto cookie
  - ☒ rescrita de URL
    - response.encodeURL ou response.encodeRedirectURL
- `getSession(false)`
  - ✓ retorna sessão existente ou null

◉ Servlet: Sessões

```
public class SessionServlet extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws IOException, ServletException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    HttpSession session = request.getSession(false);

    if (session == null)
      response.sendRedirect("ServletShopping");
    else
      cart = (ShoppingCart) session.getAttribute("ClientCart");

     items.add( cart.getItems());

    out.print("<p><a href=\"");
    out.print(response.encodeURL("/servlet/bookdetails?bookId=" + book.getBookID()));

  }
}
```

Servlet: What's new?

Servlet 3.0

- ✓ **Facilidade de desenvolvimento**
  - annotations para servlets, filters e listerners
- ✓ **Conectividade e Extensibilidade (pluggability)**
  - modularização do web.xml
    - Servlet 2.5: web.xml monolítico
    - Servler 3.0: web.xml + web-fragment.xml
  - Registro dinâmico de servlets, filters e listerners
    - adiciona dinamicamente no ServletContextListener
    - SevletContext
      - ☑ addServlet(), addFilter() e addListener()

```
public void contextInitialized(ServletContextEvent event) {
        ServletContext context = event.getServletContext();

        Dynamic dynamic = context.addServlet("ServletA", ServletA.class);
        dynamic.addMapping("/ServletA");
}
```

Servlet: What's new?

Servlet 3.0

✓ Suporte a processamento Assíncrono

- processamento síncrono
  - threads associadas a request ficam ociosas (idle state)
    - ☑ esperar recurso disponibilizar dados: query DB
    - ☑ esperar evento ocorrer: mensagem JMS
  - bloqueia a operação e consome threads e recursos
- processamento assíncrono
  - delega o código que bloqueia a operação para outra thread
  - thread que atende request volta para o container
  - aplicação web escalável
    - ☑ deve garantir que thread request não esteja ociosa

Servlet: What's new?

Servlet 3.0

✓ Suporte a processamento Assíncrono

- AsyncContext (javax.servlet)
  - define métodos para o contexto assíncrono

```
•addListener()      •dispatch()      •getTimeout()
•createListener()   •getRequest()    •setTimeout()
•complete()         •getResponse()   •start()
```

- ServletRequest.startSync
  - coloca o request em modo assíncrono
  - inicializa o contexto assíncrono (AsyncContext)
- AsyncListener
  - listener que será notificado dos eventos assíncronos

```
•onComplete()       •onStartAsync()
•onError()          •onTimeout()
```

  - AsyncEvent
- asyncSupported
  - habilita o processamento assíncrono

```
@WebServlet(asyncSupported=true)
<async-supported>true</async-supported>
```

Servlet: What's new?

```
11 @WebServlet(urlPatterns={"/asyncservlet"}, asyncSupported=true)
12 public class AsyncServlet extends HttpServlet {
13     private MyRemoteResource resource;
14
15     public void init(ServletConfig config) {
16         resource = MyRemoteResource.create("config1=x,config2=y");
17     }
18
19     protected void doGet(HttpServletRequest request, HttpServletResponse response)
20             throws ServletException, IOException {
21         response.setContentType("text/html;charset=UTF-8");
22
23         final AsyncContext acontext = request.startAsync();
24
25         acontext.start(new Runnable() {
26             public void run() {
27                 String param = acontext.getRequest().getParameter("param");
28                 String result = resource.process(param);
29                 HttpServletResponse response = (HttpServletResponse) acontext.getResponse();
30
31                 /* ... print to the response ... */
32
33                 acontext.complete();
34             }
35         });
36     }
37 }
38
```

Servlet: What's new?

Servlet 3.0

- ✓ **Melhorias de segurança**
  - HttpServletRequest
    - métodos: login(username, password) e logout()
    - autenticação programática
  - Cookies HttpOnly
    - não pode ser acessado por scripts no lado cliente
    - técnica de proteção contra cross-site scripting (XSS)
    - `javax.servlet.http.Cookie`
      - `void setHttpOnly(boolean isHttpOnly)`
      - `boolean isHttpOnly()`

Servlet: What's new?

- Servlet 3.1
  - ✓ **Non-blocking I/O**
    - I/O tradicional
      - Thread pode precisar aguardar input/ouput
    - non-blocking I/O
      - listerners (ReadListener e WriteListener)
        - request/response em método callback
      - ServletInputStream
        - isFinished, setReadListner e isReady
      - ServlerOutputStream
        - setWriteListener e isReady()

```
input.setReadListener(new ReadListener() {
    byte buffer[] = new byte[4*1024];
    StringBuilder sbuilder = new StringBuilder();
    public void onDataAvailable() {
        try {
            do {
                int length = input.read(buffer);
                sbuilder.append(new String(buffer, 0, length));
            } while(input.isReady());
        } catch (IOException ex) { ... }
    }
});
```

[19] CESPE - 2010 - MPU

Com relação à tecnologia Servlet, julgue o item subsequente.

Em um contêiner Servlet, a execução do programa MpuServlet1 a seguir implica, também, a execução do programa MpuServlet2.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MpuServlet1 extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Concurso MPU</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Concurso do MPU- Tela 1</h1>");
    response.flushBuffer();
    RequestDispatcher rd = request.getRequestDispatcher("MpuServlet2");
    rd.forward(request,response);
    out.println("</body>");
    out.println("</html>");
    out.close();
  }
}
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MpuServlet2 extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Concurso MPU</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Concurso do MPU- Tela 2</h1>");
    out.println("</body>");
    out.println("</html>");
    out.close();
  }
}
```

[20] CESGRANRIO - 2008 - TJ-RO

O método da interface javax.servlet.http.HttpSession, utilizado para finalizar uma sessão de usuário em um container J2EE, é

a) cancel()

b) delete()

c) destroy()

d) invalidate()

e) release()

[21] CESGRANRIO - 2012 - TRANSPETRO

Suponha que uma aplicação WEB construída com a linguagem Java contém uma variável de sessão que faz referência a um objeto da classe Usuario. Suponha, também, que haja nessa aplicação uma função de nome doGet, cuja assinatura é apresentada a seguir.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

Note que esse método possui um parâmetro denominado request, cuja classe é HttpServletRequest, componente da API (Application Programing Interface) de Servlets.

Suponha ainda que existe uma variável de sessão cuja referência é feita pela cadeia de caracteres "usuario".

[21] CESGRANRIO - 2012 - TRANSPETRO

Qual instrução usa corretamente o parâmetro request para atribuir o objeto armazenado na variável de sessão a uma variável de referência do tipo Usuario e denominada usr, definida no corpo da função doGet?

a) usr = (Object) request.getSession().getParameter("usuario");

b) usr = (Usuario) request.getSession().getAttribute("usuario");

c) usr = (Usuario) request.getSession().getParameter("usuario");

d) usr = request.getSession().getAttribute("usuario");

e) usr = request.getSession().getParameter("usuario");

Gabarito

[01]	E
[02]	A
[03]	E
[04]	E
[05]	A
[06]	C
[07]	E

[08]	B
[09]	E
[10]	C
[11]	D
[12]	E
[13]	A
[14]	C

[15]	B
[16]	C
[17]	A
[18]	D
[19]	E
[20]	D
[21]	B

Java Enterprise Edition

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

Leonardo Marcelino
http://www.itnerante.com.br/profile/LeonardoMarcelino