Globalcode®
The Developers Company

Open-source Education

Dicas e revisões técnicas para certificação SCJP

Iniciativa Globalcode

Palestrante

Yara M. H. Senger

yara@globalcode.com.br

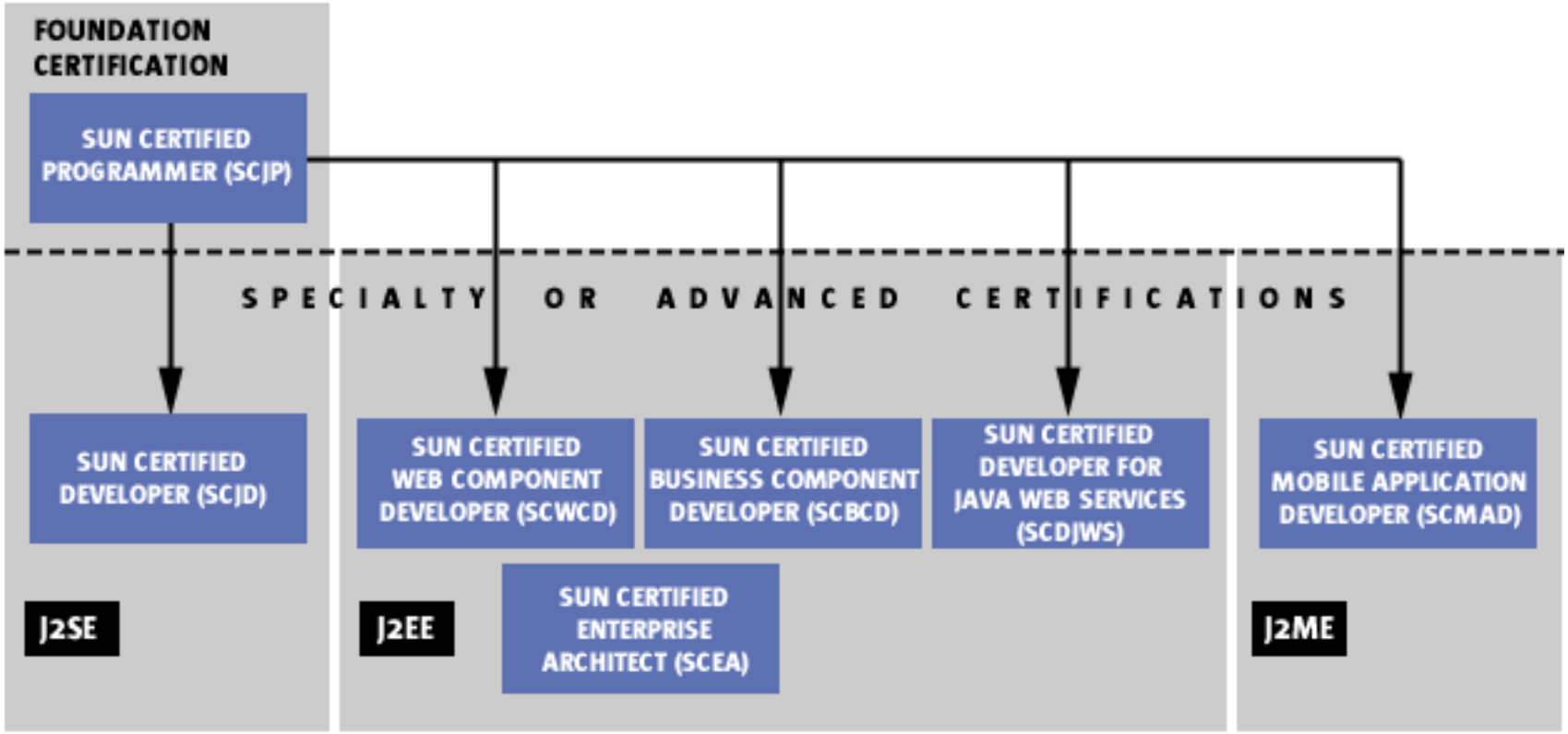
Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Certificações oficiais

- Representam uma “habilitação” internacional e oficial;
- Formação acadêmica + experiência + certificação = muitos empregos;
- Licitações frequentemente exigem um determinado número de pessoas certificadas;
- Provas são feitas em centros Prometric;
- A certificações agora são vitalícias

Certificações oficiais



Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Comprando o voucher

1) Ligar na Sun e solicitar o voucher (0800 55 7863)
2) Você receberá um boleto via e-mail em até 3 dias úteis. Atualmente o voucher custa R$ 330,00.
3) Você terá até 5 dias úteis para realizar o pagamento do boleto.
4) Você receberá o voucher pelo correio em até 15 dias úteis.
5) O voucher tem duração de 1 ano, no entanto, ele chega a Sun do Brasil com 12 meses de validade, mas até que ele chegue as nossas mãos geralmente restam entre 10 e 11 meses até que ele expire.

Agendando a prova

1) Entre no site da prometric (www.prometric.com) e esclha a opção Locate a Test Center
3) Informe o orgão certificador (Sun Microsystems) e o país.
4) Clique no link Locate a Test Center
5) Selecione o Client Sun Microsystems e o programa Sun Microsystems (310,311)
6) Selecione o exame: 310-035 Sun Certified Programmer for the Java 2 Plataform 1.4 e o idioma desejado.
7) A lista de centros prometric será apresentada
8) Escolha o centro mais próximo de você e agende sua prova pelo telefone ou no próprio site da prometric.

A prova

Pré-requisitos sugeridos: experiência de 6 meses a 1 ano

Tipo de prova: Questões de multipla escolha e respostas curtas

Número de questões: 61

Pontuação necessária: 52% (32 questões)

Duração da prova: 2 horas

A prova está disponível nos seguintes idiomas:

- Inglês
- Chinês
- Japonês
- Koreano

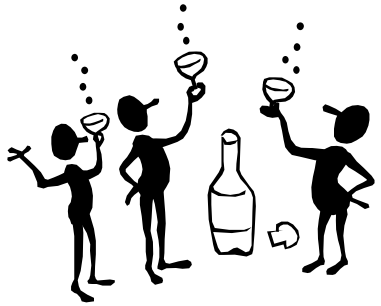
Portanto se você não sabe e não quer estudar inglês...
... já pode começar a estudar chinês, japonês ou koreano!

A prova

Antes de começar a fazer a prova você terá que responder algumas perguntas fazendo uma auto-avaliação do seu conhecimento, mas o tempo de resposta deste questionário não será subtraido dos 120 minutos que você tem para fazer a prova.

Ao terminar a prova você pode verificar sua pontuação no sistema, e receberá um documento com seu indice de acerto e a sua pontuação por tópico.

A comemoração...



Pronto, basta atualizar seu currículo e enviar um e-mail para a Globalcode dizendo que você é o mais novo profissional certificado do mercado!

Depois de 30 dias de comemoração você receberá um kit contendo:

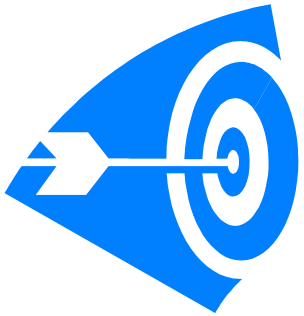
- Sua “carteirinha” de Sun Certified Programmer for Java 1.4
- O certificado
- A documentação para utilização do logo que poderá ser colocado em seu cartão de visitas ou outros documentos.

Você já pode começar a se preparar para a próxima certificação!

Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Objetivos do exame

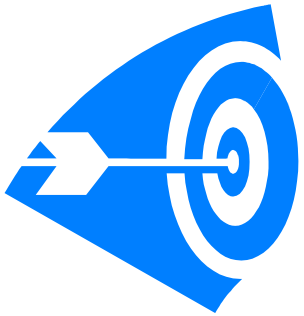


- Declaração e controle de acesso
- Controle de fluxo, assertions e tratamento de exceções
- Garbage Collection
- Fundamentos da linguagem
- Operadores e atribuições
- Threads
- Classes fundamentais do pacote java.lang
- Collections Framework

Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Objetivos do exame



Declaração e controle de acesso

- Declaração, construção e inicialização de arrays
- Declaração de classes, inner classes, métodos, atributos
- Utilização de todos os modificadores
- Construtores

Construtores

Construtores

- Toda classe tem no mínimo um construtor

- Se nenhum construtor for explicitamente declarado, então será criado implicitamente o construtor default

```
public class Classe1 {

    public static void main(String a
        Classe1 c1 = new Classe1();
    }
}
```

Apesar de não declararmos explicitamente nenhum construtor podemos utilizar o construtor default (sem argumentos)

Construtores

Construtores

- Toda classe tem no mínimo um construtor

- Se nenhum construtor for explicitamente declarado, então será criado implicitamente o construtor default

```
public class Classe1 {
    public Classe1(String param){
        System.out.println("Construt
    }
    public static void main(String a
        Classe1 c1 = new Classe1();
    }
}
```

ERRO de COMPILAÇÃO
Como foi declarado um construtor que recebe uma String o construtor default não é criado, e portanto não podemos utilizá-lo.

Construtores

Construtores

Um construtor pode chamar outros construtores de sua classe através de **this( )**, ou construtores de sua superclasse através de **super( ),** porém quando isto ocorre estas chamadas devem ocorrer na primeira linha do construtor. Desta forma, não é possível chamar ambos em um mesmo construtor;

Quando não é feita nenhuma chamada explicita ao construtor da própria classe ou a algum construtor da super-classe esta chamada é feita implicitamente invocando o construtor default da super-classe.

Construtores

Construtores

```
public class Classe1 {
    public Classe1(){
        System.out.println("Construtor da Classe1");
    }
}

public class Classe2 extends Classe1{
    public static void main(String args[]){
        Classe2 c2 = new Classe2();
    }
}
```

Qual a saída gerada quando tentamos compilar e executar a Classe2?

R: **Construtor da Classe1**, pois na Classe2 foi criado o construtor default, que por sua vez faz uma chamada ao construtor sem argumentos da super classe, que por sua vez imprime “Construtor da Classe1”

Construtores

Construtores

```
public class Classe1 {
    public Classe1(String arg){
        System.out.println("Construtor da Classe1");
    }
}
public class Classe2 extends Classe1{
    public static void main(String args[]){
        Classe2 c2 = new Classe2();
    }
}
```

Qual a saída gerada quando tentamos compilar e executar a Classe2?

R: **Erro de compilação**, pois no construtor default da Classe2 é feita uma chamada implicita ao construtor default da Classe1, que não existe já que foi declarado explicitamente um construtor que recebe uma String na Classe1. Ex:

```
public Classe2(){
   super()
}
```

Construtores

Exemplo de questão

Consider the following example:

```
class First {
public First (String s) {
   System.out.println(s);
}
}

public class Second extends First {
public static void main(String args[]){
  new Second();
}
}
```

What is the result of compiling and running the Second class?

C) Nothing happens

B) A string is printed to the standard out

C) An instance of the class First is generated

D) An instance of the class Second is created

E) An exception is raised at runtime stating that there is no null parameter constructor in class First.

F) The class second will not compile as there is no null parameter constructor in the class First

Construtores

Exemplo de questão

Consider the following example:

```
class First {
public First (String s) {
   System.out.println(s);
}
}

public class Second extends First {
public static void main(String args[]){
  new Second();
}
}
```

What is the result of compiling and running the Second class?

C) Nothing happens

B) A string is printed to the standard out

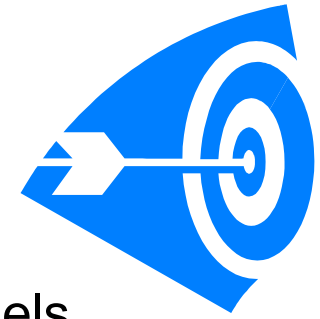
C) An instance of the class First is generated

D) An instance of the class Second is created

E) An exception is raised at runtime stating that there is no null parameter constructor in class First.

F) The class second will not compile as there is no null parameter constructor in the class First

Objetivos do exame



Controle de fluxo, assertions e tratamento de exceções

- Utilização de if, switch, for, while, break, continue, labels

- try, catch,finally, throw e throws

- Tratamento de erros com override, overloading, interfaces e métodos abstratos

- Utilização de assertions e quando deve ou não ser utilizado

Tratamento de erros

Tratamento de erros com override, overloading, interfaces e métodos abstratos

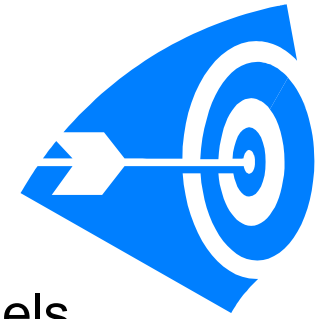
Quando fazemos override de um método ou implementamos um método abstrato de uma classe ou interface não é permitido lançar mais exceções do que o método da super classe ou interface declara, a menos é claro que seja uma RuntimeException.

Tratamento de erros

Portanto, o código abaixo não compila, pois o metodo1, sobrescrito na ClasseA lança uma exceção do tipo MinhaException enquanto o método original não lança nenhuma excpetion!

```
public class Super {
    public void metodo1(){
       System.out.println("Este metodo não lança nenhuma excecao");
    }
}

public class ClasseA extends Super {

   public void metodo1() throws MinhaException{
        System.out.println( "Este método pode lançar uma MinhaExcecao");
    }
}
```

Objetivos do exame



Controle de fluxo, assertions e tratamento de exceções

- Utilização de if, switch, for, while, break, continue, labels

- try, catch,finally, throw e throws

- Tratamento de erros com override, overloading, interfaces e métodos abstratos

- Utilização de assertions e quando deve ou não ser utilizado

Assertions

O que é assertion?

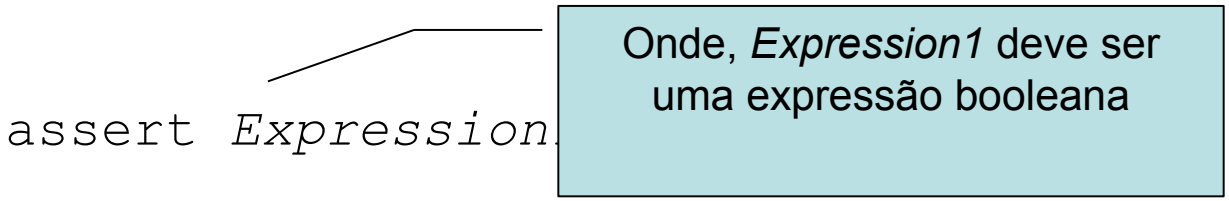
- Uma forma de garantir que determinada condição é verdadeira
- Uma verificação de uma condição que pode ser ativada ou destaivada
- Um recurso inserido a partir do JDK1.4

assert

- A palavra assert é reservada se assertions estiver ativado, pois código utilizando assertions não pode ser executado em versões anteriores de JDK

Assertions

Utilização de assertions



Se a `Expression1` for falsa, então será lançado um AssertionError sem nenhuma mensagem detalhada.

Assertions

Exemplo de utilização de assertions

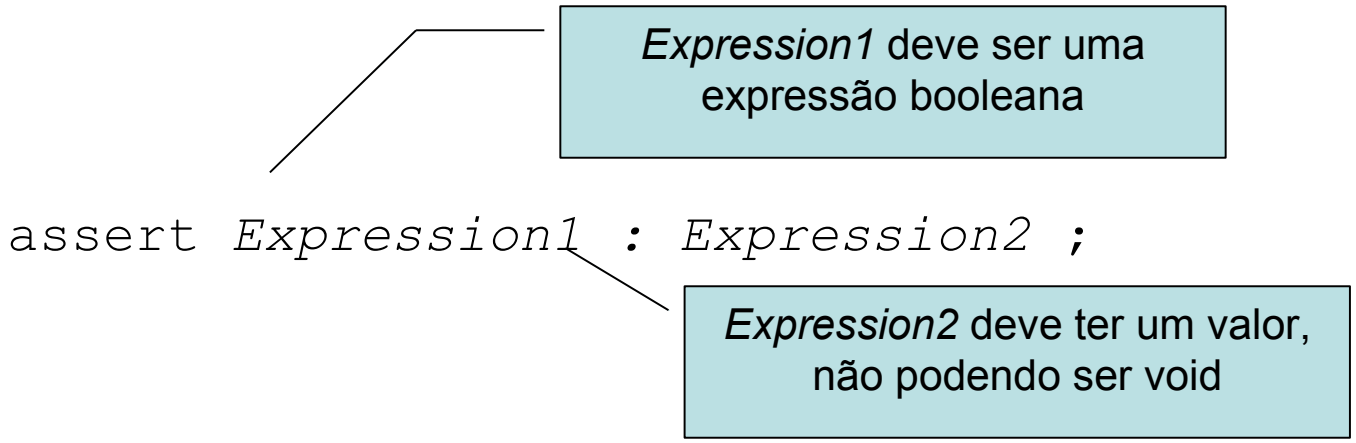
```
public class AssertTest {
    public void methodA(int i){
        assert i >=0;
        System.out.println(i);
    }

    public static void main(String args[]){
        AssertTest test = new AssertTest();
        test.methodA(-10);
    }

}
```

Se i for menor ou igual a zero então será lançado um AssertionErro

Neste exemplo o valor passado é menor do que 0, ou seja, neste ponto receberemos um AssertionError.

Assertions

Utilização de assertions



Se a *Expression1* for falsa, então será lançado um AssertionError com a mensagem detalhada criada a partir da representação em String da *Expression2*.

Assertions

Compilando classes que utilizam assertions

- Para habilitar o uso de assertions devemos compilar as classes utilizando o comando **javac – source 1.4**

- Para ignorar assertions compilamos com o comando javac – source 1.3 (default)

Assertions

Executando classes que utilizam assertions

Para executar uma classe com assertions podemos utilizar as diretivas de linha de comando resumidas na tabela abaixo:

Diretiva	Exemplo	Descrição
-ea	java -ea	Habilita assertions
-da	java -da	Desabilita assertions
-ea <NomeClasse>	java -ea:Foo	Habilita assertions para classe Foo
-da <NomeClasse>	java -da:Foo	Desabilita assertions para classe Foo
-ea <nomePacote>...	java - ea:com.globalcode...	Habilita assertions para o pacote com.globalcode
-da <nomePacote>...	java - da:com.globalcode...	Desabilita assertions para o pacote com.globalcode
-esa	java -esa	Habilita assertions para as classes sistema
-dsa	java -dsa	Desabilita assertions para as classes sistema

Assertions

Quando devemos utilizar assertions?

Situações onde determinada condição não faz sentido
e não deveria acontecer, como por exemplo um switch
onde acreditamos cobrir todas as possiblidades com case e não
programamos nada no default.

Exemplo:

```
switch(diaSemana) {
 case diaSemana.SEGUNDA: ... break;
 case diaSemana.TERCA  : ... break;
 case diaSemana.QUARTA : ... break;
 case diaSemana.QUINTA : ... break;
 case diaSemana.SEXTA  : ... break;
 case diaSemana.SABADO : ... break;
 case diaSemana.DOMINGO: ... break;
}
```

Assertions

- Com assertions deveria ficar da seguinte forma:

```
switch(diaSemana) {
 case diaSemana.SEGUNDA: ... break;
 case diaSemana.TERCA  : ... break;
 case diaSemana.QUARTA : ... break;
 case diaSemana.QUINTA : ... break;
 case diaSemana.SEXTA  : ... break;
 case diaSemana.SABADO : ... break;
 case diaSemana.DOMINGO: ... break;
 default:
assert false:diaSemana;

}
```

Assertions

Quando devemos utilizar assertions?

- Podemos utilizar assertions em um lugar que não deveria ser alcançado.

- Pré-condições : podemos utilizar assertions para testar parâmetros de métodos não-públicos, ou seja, métodos que devem ser independentes dos dados informados pelos usuários.

- Pós-condições: podemos utilizar assertions para testar pós-condições em métodos públicos ou não.

Assertions

Quando NÃO devemos utilizar assertions?

- Não devemos utilizar assertions para checagem de parâmetros de métodos públicos, visto que assertions pode ser desabilitada e queremos que os parâmetros sejam sempre checados.

- Não utilize assertions para realizar nenhuma tarefa essencial para o funcionamento da aplicação, visto que assertions pode ser desabilitada.

Assertions

Exemplo de questão

```
class A {
  void m1(int i) {
    int j = i % 3;
    switch (j) {
      case 0 :
        System.out.print("0");
        break;
      case 1 :
        System.out.print("1");
        break;
      default :
        assert j == 2;
        System.out.print(j);
    }
  }
  public static void main(String[] args){
    A a = new A();
    for (int i = 5; i >= -1; i--) {
      a.m1(i);
    }
  }
}_
```

A) With assertions enabled it prints "210210-1" followed by an AssertionError message.

B) With assertions enabled it prints "210210" followed by an AssertionError message.

C) With assertions enabled it prints only "210210".

D) With assertions enabled it prints nothing.

E) With assertions disabled it prints "210210-1“

F) With assertions disabled it prints only "210210“

G) Assertions should not be used within the default case of a switch statement.

H) A compiler error is generated

I) None of the above

Assertions

Exemplo de questão

```
class A {
  void m1(int i) {
    int j = i % 3;
    switch (j) {
      case 0 :
        System.out.print("0");
        break;
      case 1 :
        System.out.print("1");
        break;
      default :
        assert j == 2;
        System.out.print(j);
    }
  }
  public static void main(String[] args) {
    A a = new A();
    for (int i = 5; i >= -1; i--) {
      a.m1(i);
    }
  }
}_
```

i	j	saida
5	2	2
4	1	1
3	0	0
2	2	2
1	1	1
0	0	0
-1	-1	?

Com assertions: 210210 seguido de AssertionError
Sem assertions: 210210-1

Assertions

Exemplo de questão

```
class A {
  void m1(int i) {
    int j = i % 3;
    switch (j) {
      case 0 :
        System.out.print("0");
        break;
      case 1 :
        System.out.print("1");
        break;
      default :
        assert j == 2;
        System.out.print(j);
    }
  }
  public static void main(String[] args){
    A a = new A();
    for (int i = 5; i >= -1; i--) {
      a.m1(i);
    }
  }
}_
```

A) With assertions enabled it prints "210210-1" followed by an AssertionError message.

B) With assertions enabled it prints "210210" followed by an AssertionError message.

C) With assertions enabled it prints only "210210".

D) With assertions enabled it prints nothing.

E) With assertions disabled it prints "210210-1“

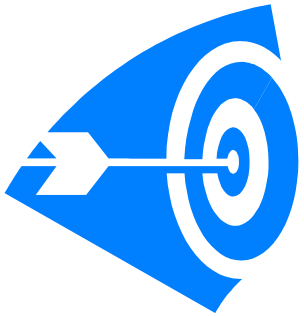
F) With assertions disabled it prints only "210210“

G) Assertions should not be used within the default case of a switch statement.

H) A compiler error is generated

I) None of the above

Objetivos do exame



Garbage Collection

- Garantias do mecanismos de coleta de lixo

- Escrever código que explicitamente faça objetos elegíveis para ser coletado pelo Garbage Colector.

- Reconhecer o ponto no código onde os objetos tornam-se elegíveis para serem coletados.

Garbage Collection

Reconhecer o ponto no código onde os objetos tornam-se elegíveis para serm coletados

Exemplo de questão

Objetivos do exame

Fundamentos da linguagem

- Identificar a declaração correta de pacotes, import, classes,métodos, variáveis e atributos, além do método main

- Identificar a implentação correta de interfaces (inclusive a interface Runnable)

- Passagem de parâmetros para o método main

- Identificar todas as palavras reservadas da linguagem

- Conhecer o tamanho das variáveis de cada tipo primitivo e os valores (literais) de cada tipo

Palavras reservadas

Lista de palavras reservadas

abstract	const	final	instanceof	private	switch	void
boolean	continue	finally	int	protected	synchonized	volatile
break	default	float	interface	public	this	while
byte	do	for	long	return	throw	
case	double	goto	native	short	throws	
catch	else	if	new	static	transient	
char	extends	implements	null	strictfp	true	
class	false	import	package	super	try	

Palavras reservadas

Exemplo de questão

Which of the following are reserved words in Java?

```
A) Class
B) goto
C) continue
D) const
E) sizeof
F) include
```

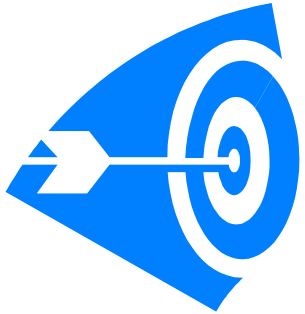
Palavras reservadas

Exemplo de questão

Which of the following are reserved words in Java?

A) Class
B) goto
C) continue
D) const
E) sizeof
F) include

Objetivos do exame



Operadores e atribuições

- Conhecer os beneficios obtidos na utilização de encapsulamento

- Implementar o encapsulamento

- Conhecer o relacionamento “tem um” e “é um”

- Escrever código que invoque métodos sobrescritos, sobrecarregados, e construtores

- Instanciar objetos de classes concretas, inclusive inner classes

Override

Escrever código que invoque métodos sobrescritos

```
public class Super {
    public void metodo1(){
       System.out.println("Método1 da classe Super");
    }
}

public class ClasseA extends Super {
    public void metodo1() {
        System.out.println( "Método1 da ClasseA");
    }
}
public class Teste {
    public static void main(String[] args) {
        Super s = new ClasseA();
        s.metodo1();
    }
}
```

O que será impresso?

Resposta: “Método1 da ClasseA”

Override

Exemplo de questão

Given the following classes:

```
class Vehicle {
    public void drive() {
        System.out.println("Vehicle: drive");
    }
}
class Car extends Vehicle {
    public void drive() {
        System.out.println("Car: drive");
    }
}
public class Test {
    public static void main (String args []) {
        Vehicle v; //line 1
        Car c;     //line 2
        v = new Vehicle(); //line 3
        c = new Car();     //line 4
        v.drive();         //line 5
        c.drive();         //line 6
        v = c;             //line 7
        v.drive();         //line 8
    }
}_
```

What will be the effect of compiling and running class Test?

A) Generates a compiler error at line 7;
B) Generates runtime error at line 7;
C) # Prints out:
Vehicle: drive
Car: drive
Car: drive
D) # Prints out:
Vehicle: drive
Car: drive
Vehicle: drive

Override

Exemplo de questão

Given the following classes:

```
class Vehicle {
    public void drive() {
        System.out.println("Vehicle: drive");
    }
}
class Car extends Vehicle {
    public void drive() {
        System.out.println("Car: drive");
    }
}
public class Test {
    public static void main (String args []) {
        Vehicle v; //line 1
        Car c;     //line 2
        v = new Vehicle(); //line 3
        c = new Car();     //line 4
        v.drive();         //line 5
        c.drive();         //line 6
        v = c;             //line 7
        v.drive();         //line 8
    }
}
```

What will be the effect of compiling and running class Test?

A) Generates a compiler error at line 7;

B) Generates runtime error at line 7;

C) # Prints out:
Vehicle: drive
Car: drive
Car: drive

D) # Prints out:
Vehicle: drive
Car: drive
Vehicle: drive

Objetivos do exame

Operadores e atribuições

- Conhecer os beneficios obtidos na utilização de encapsulamento

- Implementar o encapsulamento

- Conhecer o relacionamento “tem um” e “é um”

- Escrever código que invoque métodos sobrescritos, sobrecarregados, construtores sobrescritos ou da super-classe

- Instanciar objetos de classes concretas, inclusive inner classes

Inner class

Instanciar objetos de classes concretas, inclusive inner classes

```
public class OuterClass {
    public class InnerClass{
        public void teste(){
            System.out.println("Testando a InnerClass");
        }
    }
}
public class TesteInner {

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.teste();
    }
}
```

Objetivos do exame

Threads

- Definir, instanciar e iniciar a execução de threads utilizando a classe Thread ou a interface Runnable

- Conhecer as condições que inibem a execução de uma thread

- Escrever código utilizando wait, notify e notifyAll

Threads

A funcionalidade de uma thread deve ser programada no **método run**(), que não recebe nenhum parâmetro, não lança nenhuma exception e retorna void.

- Existem duas maneiras de criarmos uma classe que poderá ser executada em uma thread separada pela Virtural Machine:

  1) Estender a classe Thread e sobrescrever o método run().

  2) Criar uma classe que implementa a interface Runnable e implementar o método run();

Threads

1) Estendendo a classe Thread

A classe Thread tem um método **start()**, que deve ser chamado para registrar a thread no thread scheduler do sistema operacional, se chamarmos o método run diretamente ele será executado na mesma thread do método que o chama e não em uma separada.

```
public class MinhaThread extends Thread {
    public void run() {
        System.out.println("Coloque aqui a tarefa");
    }
}
public class TesteThread{
    public static void main(String args[]){
        MinhaThread t = new MinhaThread();
        t.start();
    }
}
```

Threads

2) Implementando a interface Runnable

Para implementarmos a interface Runnable é necessário implementar o método run().

Para colocarmos a nossa classe para ser executada em uma Thread separada devemos criar uma instância da classe Thread passando como parâmetro no construtor a nossa instância de Runnable.

```
public class MinhaRunnable implements Runnable {
    public void run() {
        System.out.println("Coloque aqui a tarefa");
    }
}
public class TesteRunnable{
    public static void main(String args[]){
        MinhaRunnable r = new MinhaRunnable();
        Thread t = new Thread(r);
        t.start();
    }
}
```

Threads

Exemplo de questão

Which one statement below is true concerning the following code?

```
1.class Greebo extends java.util.Vector implements Runnable   {
2.  public void run(String message) {
3.   System.out.println("in run() method:  " + message);
4.  }
5.}
6.
7. class GreeboTest {
8.   public static void main(String args[])  {
9.       Greebo g = new Greebo();
10.      Thread t = new Thread(g);
11.    t.start();
12.   }
13.  }
```

A) There will be a compiler error, because class Greebo does not implement the Runnable interface.
B) There will be a compiler error at line 10, because you cannot pass a parameter to the constructor of a Thread.
C) The code will compile correctly but will crash with an exception at line 10.
D) The code will compile correctly but will crash with an exception at line 11.
E) The code will compile correctly and will execute without throwing any exceptions_

Threads

Exemplo de questão

Which one statement below is true concerning the following code?

```
1.class Greebo extends java.util.Vector implements Runnable   {
2.  public void run(String message) {
3.   System.out.println("in run() method:  " + message);
4.  }
5.}
6.
7. class GreeboTest {
8.   public static void main(String args[])  {
9.       Greebo g = new Greebo();
10.      Thread t = new Thread(g);
11.    t.start();
12.   }
13.  }
```

A) There will be a compiler error, because class Greebo does not implement the Runnable interface.
B) There will be a compiler error at line 10, because you cannot pass a parameter to the constructor of a Thread.
C) The code will compile correctly but will crash with an exception at line 10.
D) The code will compile correctly but will crash with an exception at line 11.
E) The code will compile correctly and will execute without throwing any exceptions_

Objetivos do exame

Classes fundamentais do pacote java.lang

- Conhecer os seguintes métodos da classe Math
abs, ceil, floor, max, min, random, round, sin, cos, tan, sqrt.

- String

- Entender o objetivo das Wrapper classes, sua utilização e os seguintes métodos:
doubleValue, floatValue, intValue , longValue , parseXxx , getXxx, toString, toHexString

Inner class

String

- Strings são objetos imutáveis, ou seja, uma vez construídos não podem ter seus valores alterados.

- Nenhum método da classe String altera a String, apenas criam uma nova String

- A JVM mantém um “pool” de Strings de forma que literais com o mesmo valor só são criadas uma única vez, sendo que todas as referências a seus conteúdos apontam para o mesmo endereço (Ex: “John” == “John” retorna true);

Inner class

Exemplo de questão (short answer)

What are the contents of String a after the following lines of code are executed?

```
String a = "pay";
String b= "lay";
a.concat("2");
a.replace('p','d');
b="no";
a+=b.toUpperCase();
```

RESPOSTA: _____

Inner class

Exemplo de questão (short answer)

What are the contents of String a after the following lines of code are executed?

```
String a = "pay";
String b= "lay";
a.concat("2");
a.replace('p','d');
b="no";
a+=b.toUpperCase();
```

RESPOSTA: payNO

Objetivos do exame

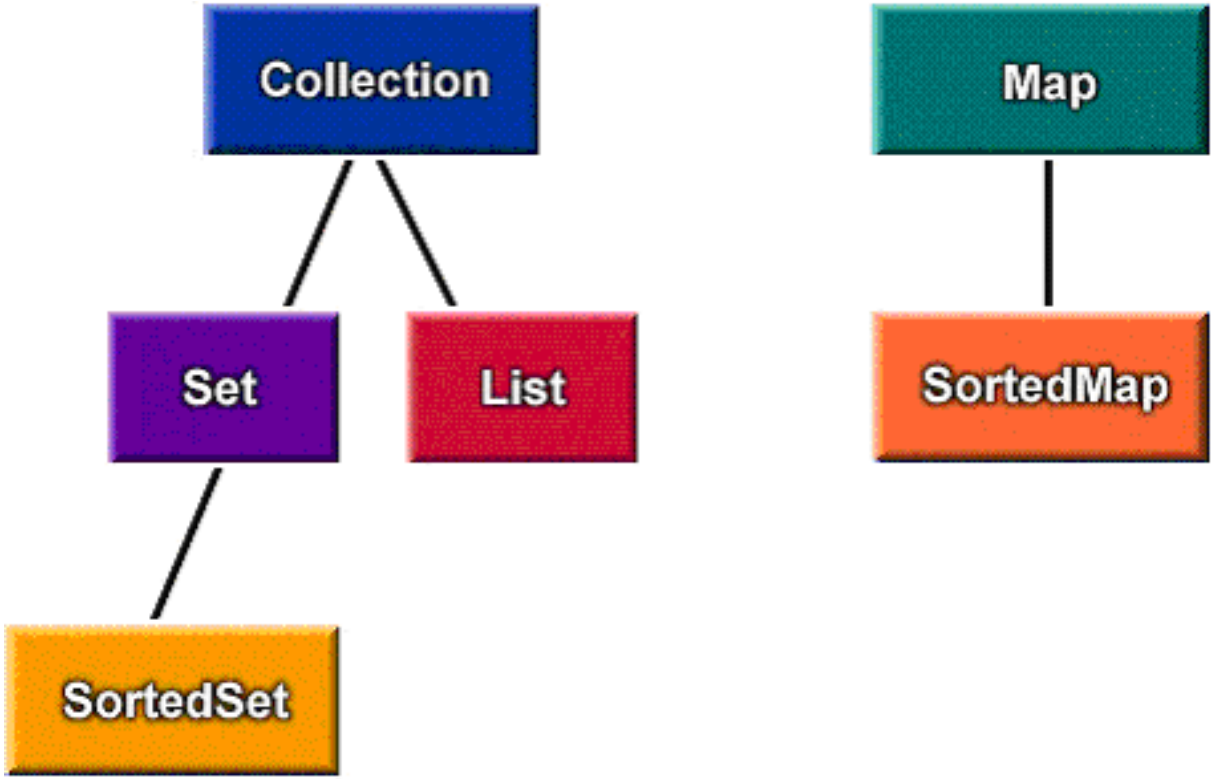
Collections Framework

- Reconhecer quais coleções devem ser utilizadas para atender determinados requisitos

- Identificar implementações corretas ou incorretas do método hashCode()

Objetivos do exame

Collections Framework



Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão da sintaxe da linguagem
5. Revisão de Orientação a Objetos
6. Revisão sobre Threads
7. Pegadinhas
8. Simulado
9. Correção do simulado
10. Certificações Java & Oportunidades de trabalho
11. Proposta de plano de estudos

Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Agenda

Tempo para resolução do simulado

Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Agenda

Tempo para correção do simulado

Agenda

1. Certificações oficiais
2. Processo de certificação SCJP
3. Objetivos do exame
4. Revisão técnica e pegadinhas
5. Simulado
6. Correção do simulado
7. Proposta de plano de estudos

Plano de estudos

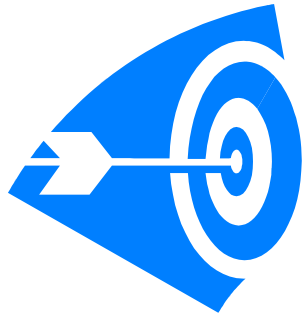
Plano I: Para aqueles que já trabalham ou conhecem bem Java

Declaração e controle de acesso	4
Controle de fluxo, assertions e tratamento de exceções	12
Garbage Collection	4
Fundamentos da linguagem	16
Operadores e atribuições	6
Threads	16
Classes fundamentais do pacote java.lang	4
Collections Framwork	10
TOTAL	74

Plano de estudos

Plano II: Para aqueles que conhecem razoavelmente Java, mas ainda não trabalham com a tecnologia

Declaração e controle de acesso	8
Controle de fluxo, assertions e tratamento de exceções	24
Garbage Collection	8
Fundamentos da linguagem	32
Operadores e atribuições	12
Threads	32
Classes fundamentais do pacote java.lang	8
Collections Framwork	20
TOTAL	148



Links úteis

http://suned.sun.com/US/certification/java/index.html : site oficial sobre as certificações Java

http://suned.sun.com/US/certification/resources/epractice.html : site com simulados para a certificação, alguns são gratuitos e outros pagos.

http://www.prometric.com : site oficial prometric,
onde é possível encontrar os centros autorizados e agendar a prova.

http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html: Material adicional sobre assertions

Links para sites com simulados

http://www.danchisholm.net

http://javacertificate.com

http://www.whizlabs.com

http://www.javaranch.com/