**[Ir para o conteúdo principal](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/quiz/qyOhh/sobre-tipos-genericos/attempt" \l "main)**

Parte superior do formulário

PESQUISAR NO CURSO

Pesquisar

Parte inferior do formulário

* Português (Brasil)
* **H**

Heleno Cardoso da Silva Filho

Ocultar menu

* **Introdução e Visão Geral do Curso**
* **Aplicações Web com Java Servlets**
* **Tipos Genéricos em Java**

**[Vídeo:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/aFs77/introducao-a-tipos-genericos)**[VídeoIntrodução a Tipos Genéricos](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/aFs77/introducao-a-tipos-genericos)

[. Duration: 7 minutes7 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/aFs77/introducao-a-tipos-genericos)

**[Leitura:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/supplement/ImCvX/restringindo-tipos-genericos)**[LeituraRestringindo Tipos Genéricos](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/supplement/ImCvX/restringindo-tipos-genericos)

[. Duration: 10 minutes10 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/supplement/ImCvX/restringindo-tipos-genericos)

**[Vídeo:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/xKcSR/tipos-invariantes-e-covariantes)**[VídeoTipos Invariantes e Covariantes](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/xKcSR/tipos-invariantes-e-covariantes)

[. Duration: 8 minutes8 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/xKcSR/tipos-invariantes-e-covariantes)

**[Vídeo:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/e5EoC/tipos-genericos-em-metodos)**[VídeoTipos Genéricos em Métodos](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/e5EoC/tipos-genericos-em-metodos)

[. Duration: 5 minutes5 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/e5EoC/tipos-genericos-em-metodos)

**[Vídeo:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/A3CGf/quando-utilizar-tipos-genericos)**[VídeoQuando utilizar tipos genéricos?](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/A3CGf/quando-utilizar-tipos-genericos)

[. Duration: 8 minutes8 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/A3CGf/quando-utilizar-tipos-genericos)

**[Vídeo:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/svA0h/hands-on-tipos-genericos-na-pratica)**[VídeoHands-on: Tipos Genéricos na Prática](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/svA0h/hands-on-tipos-genericos-na-pratica)

[. Duration: 16 minutes16 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/svA0h/hands-on-tipos-genericos-na-pratica)

**[Teste para praticar:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/quiz/qyOhh/sobre-tipos-genericos)**[Sobre Tipos Genéricos](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/quiz/qyOhh/sobre-tipos-genericos)

[. Duration: 20 minutes20 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/quiz/qyOhh/sobre-tipos-genericos)

**[Teste:](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/exam/pbw25/voce-entendeu-como-funcionam-os-tipos-genericos)**[Você entendeu como funcionam os tipos genéricos?](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/exam/pbw25/voce-entendeu-como-funcionam-os-tipos-genericos)

[. Duration: 30 minutes30 min](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/exam/pbw25/voce-entendeu-como-funcionam-os-tipos-genericos)

* **Criando a primeira aplicação Web em Java**

1. [Semana 1](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/home/week/1)
2. Sobre Tipos Genéricos

[Anterior](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/lecture/svA0h/hands-on-tipos-genericos-na-pratica)[Próximo](https://www.coursera.org/learn/desenvolvimento-agil-com-java-avancado/exam/pbw25/voce-entendeu-como-funcionam-os-tipos-genericos)

**Sobre Tipos Genéricos**

**Envie sua tarefa**

Retomar a tarefa

**Receber nota**

Para ser aprovado 80% ou superior

**Sua nota**

-Não disponível

Curtir

Não curtir

Relatar problema

****

Voltar

**Sobre Tipos Genéricos**

Teste para praticar. • 20 min. • 11 pontos totais disponíveis.11 pontos totais

Português (Brasil)

**1.**

Pergunta 1

Qual é o par de símbolos que identifica um tipo genérico (generics)?

1 ponto

< >

! >

/ /

< !

**2.**

Pergunta 2

Para que servem os tipos genéricos?

1 ponto

Verificar com mais segurança os tipos em tempo de compilação

Diminuir a legibilidade do código da aplicação

Diminuir a quantidade de objetos no código da aplicação

Eliminar a necessidade de conversões (casting) de tipos de objetos de coleções

**3.**

Pergunta 3

O significado da seguinte declaração:

1

ArrayList<Pessoa> pessoas;

é "*A variável de instância 'pessoas' corresponde a um conjunto de objetos da classe Pessoa armazenado num objeto ArrayList*"!

1 ponto

Verdadeiro

Falso

**4.**

Pergunta 4

ArrayList<Pessoa> é uma invocação de qual declaração de tipo genérico?

1 ponto

public<E> class ArrayList

public class ArrayList<E>

public class<E> ArrayList

**5.**

Pergunta 5

Dada a seguinte classe genérica ou parametrizada:

5

}

São formas corretas de invocação de objetos dessa classe:

1 ponto

Cesta<Gato> gatos;

Cesta<Fruta> frutas;

Cesta Pirulito <pirulitos>;

public SaladaFrutas criarSalada(Cesta<Fruta> c){...}

**6.**

Pergunta 6

[Curiosidade com o objetivo de estender o conhecimento sobre tipos genéricos!]

O método seguinte compila corretamente, pois ele usa "E" para simbolizar um tipo genérico, como se encontra no tutorial da Oracle sobre Java 8:

public static <E> void imprimeArray(E[] array) {

        for (E elemento : array)

            System.out.print(elemento + " ");

        System.out.println();

}

E este aqui, compila corretamente também?

5

public static <Z> void imprimeArray(Z[] array) {

        for (Z elemento : array)

            System.out.print(elemento + " ");

        System.out.println();

}

1 ponto

Não

Sim

O par "<>" é chamado de operador diamante. Usualmente usa-se "E", simbolizando um Element of type, como no tutorial da Oracle, **<E>** ;ou muitas vezes, usa-se "T" para Type,**<T>** . Contudo, dentro do operador diamante, você pode colocar qualquer identificador que comece por letra, maiúscula ou minúscula, e qualquer número de caracteres maiúsculos ou minúsculos ou dígitos: <W>, <sTore023>, etc. Por exemplo, <Integer> é válido, mas não simboliza o Integer usual de Java; quando dentro do operador diamante, significa apenas um tipo genérico definido pelo desenvolvedor. Mas não devemos complicar o código desnecessariamente: adote a convenção usada nos documentos da Oracle sobre generics, a saber, "E" (Elemento), "N" (Número), "T" (Tipo), "K" (Chave) e "V" (Valor)!

**7.**

Pergunta 7

Indique a definição de tipo genérico errônea:

1 ponto

3

public <E> class Cesta{

   // ...

}

1

2

3

public class Cesta<E>{

   // ...

}

3

1

2

}

public <E> boolean comparar(E o1, E o2){

  // ...

3

}

public <E> E primeiro(List<E> lista){

  // ...

}

Ver a aula Entendendo Tipos Genéricos e Tipos Genéricos em Métodos

**8.**

Pergunta 8

Suponha que Fruta seja subclasse de Vegetal. Pelo conceito de Invariante, vimos que Cesta<Fruta> não é subclasse de Cesta<Vegetal>, certo? Diga, então, de que classe as classes Cesta<Fruta> e Cesta<Vegetal> estendem ou são subclasses:

1 ponto

Classe Object do Java

Classe Cesta<SerVivo>

Classe Vegetal

Classe SerVivo

**9.**

Pergunta 9

**[Responda sem rodar o código abaixo na primeira rodada da solução do quiz; depois rode o código para consolidar o seu conhecimento do assunto]**

Dado que ArrayList<E> é subclasse de List<E> que é subclasse de Collection<E>, explique por que print(li) funciona na classe abaixo:

18

15

16

17

12

13

14

10

11

8

9

6

7

3

4

5

1

2

}

        // print(al2); // erro de compilação

    }

        Collection<Integer> al2 = new ArrayList<>();

        print(li);

        print(al1);

        List<Number> li = Arrays.asList(3, 4, 6, 8, 11, 15, 28, 32);

        ArrayList<Integer> al1 = new ArrayList<Integer>(Arrays.asList(3, 4, 6, 8, 10, 11, 15, 28, 32));

    public static void main(String[] args) {

        System.out.println();

    }

    public static void print(List<? extends Number> list) {

        for (Number n : list)

            System.out.print(n + " ");

public class ExemploGenerics {

Saída:

2

3 4 6 8 10 11 15 28 32

1 ponto

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser List<Number> ou List<Integer>!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Number>!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Integer>!

**10.**

Pergunta 10

**[Responda sem rodar o código abaixo na primeira rodada da solução do quiz; depois rode o código para consolidar o seu conhecimento do assunto]**

Dado que ArrayList<E> é subclasse de List<E> que é subclasse de Collection<E>, explique por que print(al1) funciona na classe abaixo:

18

public class ExGenericsCompila {

    public static void print(List<? extends Number> list) {

        for (Number n : list)

            System.out.print(n + " ");

        System.out.println();

    }

    public static void main(String[] args) {

        List<Number> li = Arrays.asList(3, 4, 6, 8, 11, 15, 28, 32);

        ArrayList<Integer> al1 = new ArrayList<Integer>(Arrays.asList(3, 4, 6, 8, 10, 11, 15, 28, 32));

        Collection<Integer> al2 = new ArrayList<>();

        print(li);

        print(al1);

        // print(al2); // erro de compilação

    }

}

Saída:

2

1

3 4 6 8 10 11 15 28 32

3 4 6 8 11 15 28 32

1 ponto

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser List<Number> ou List<Integer>! Sendo List<Integer>, como ArrayList<E> é subclasse de List<E>, E = Integer, então ArrayList<Integer> funciona!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Integer>! Dessa forma, como ArrayList<Integer> é subclasse de List<Integer>, ArrayList<Integer> funciona!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Number>! Dessa forma, como ArrayList<Integer> é subclasse de List<Number>, ArrayList<Integer> funciona!

Correto

De fato, o parâmetro real poderá ser List<Number> ou List<Integer> ou List<qualquer outra subclasse de Number: Double, Float, BigInteger etc.>! Ver a videoaula Tipos Invariantes e Covariantes

**11.**

Pergunta 11

**[Responda sem rodar o código abaixo na primeira rodada da solução do quiz; depois rode o código para consolidar o seu conhecimento do assunto]**

Dado que ArrayList<E> é subclasse de List<E> que é subclasse de Collection<E>, explique por que print(al2) **não** compila na classe abaixo:

19

 public class ExGenericsCompila {

    public static void print(List<? extends Number> list) {

        for (Number n : list)

            System.out.print(n + " ");

        System.out.println();

    }

    public static void main(String[] args) {

        List<Number> li = Arrays.asList(3, 4, 6, 8, 11, 15, 28, 32);

        ArrayList<Integer> al1 = new ArrayList<Integer>(Arrays.asList(3, 4, 6, 8, 10, 11, 15, 28, 32));

        Collection<Integer> al2 = new ArrayList<>();

        print(li);

        print(al1);

        // print(al2); // erro de compilação

    }

}

Saída:

1

2

3 4 6 8 11 15 28 32

3 4 6 8 10 11 15 28 32

1 ponto

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Number>! Bastaria então trocar Number por Collection em print(List<? extends Collection> list)!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser List<Number> ou List<Integer>! Bastaria então trocar List por Collection em print(Collection<? extends Number> list)!

O coringa (wildcard) do método parametrizado ou genérico print(List<? extends Number> list) indica que o parâmetro real poderá ser apenas List<Integer>! Bastaria então trocar List por Collection e Number por Integer em print(Collection<? extends Integer> list)!

Correto

De fato, o parâmetro real poderá ser List<Number> ou List<Integer> ou List<qualquer outra subclasse de Number: Double, Float, BigInteger etc.>! Ver a videoaula Tipos Invariantes e Covariantes

Código de Honra do Coursera  [Saiba mais](https://learner.coursera.help/hc/articles/209818863)

Eu compreendo que enviar o trabalho de outra pessoa como se fosse meu pode resultar em reprovação permanente desse curso ou em desativação da minha conta do Coursera.

Use o nome conforme consta em seu documento oficial de identificação

EnviarSalvar esboço

Curtir

Não curtir

Relatar problema