Generics in Java

Generics in Java ?

A programação Java Generics é introduzida no J2SE 5 para lidar com objetos de tipo seguro. Torna o código estável, detectando os erros em tempo de compilação.

Antes dos genéricos, podemos armazenar qualquer tipo de objeto na coleção, ou seja, não genérico. Agora, os genéricos forçam o programador java a armazenar um tipo específico de objetos.

Generics é um termo que denota e seta na linguagem relatada uma definição e seu uso com Tipos Genéricos e Métodos.

Precisamos Genericos?

Generics

Vamos imaginar um cenário em que queremos criar uma lista em Java para armazenar Inteiro; podemos escrever:

```
List list = new LinkedList();
list.add(new Integer(1));
Integer i = list.iterator().next();
```

Surpreendentemente, o compilador reclamará da última linha. Ele não sabe que tipo de dados é retornado. O compilador exigirá conversão explícita:

```
Integer i = (Integer) list.iterator.next();
```

Não há contrato que garanta que o tipo de retorno da lista seja número inteiro. A lista definida pode conter qualquer objeto. Só sabemos que estamos recuperando uma lista inspecionando o contexto. Ao examinar tipos, ele pode garantir apenas que é um Objeto, portanto, exige uma conversão explícita para garantir que o tipo seja seguro.

Generics in Java

É um perigo em potencial para uma **ClassCastException**

Torna nossos códigos mais poluídos e menos legíveis

Destrói benefícios de uma linguagem com **tipos fortemente definidos**

Permite que uma única classe trabalhe com uma grande **variedade de tipos**

É uma forma natural de eliminar a necessidade de se fazer **cast**

Preserva benefícios da checagem de tipos

Vantagens

Generics

Reusabilidade

Tipos Seguros

Individual Cast não é Obrigatório

Implementando non genericos algoritmos

Types

Generics

Generic Types Interface

```
interface GenericInterface<T1, T2> {
         T1 PerformExecution(T1 x);
         T2 PerformExecution(T2 x);
        }

class GenericClass implements GenericInterface<String,
Integer> {

        public Integer PerformExecution(String x) {
                //code
        }
        public String PerformExecution(Integer x) {
                //code
        }

}
```

Types

Generics

Generic Types Class

```
class GenericClass {
        private Object x;
        public void set(Object x) {this.x = x;}
        public Object get() { return x;}
}
```

Types

Generics

Generic Types Method

```
public <T> List<T> fromArrayToList(T[] a) {
    return Arrays.stream(a).collect(Collectors.toList());
}
```

Types

Generics

Generic Types Constructor

```
class Dimension<T> {
        private T  length;
        private T  width;
        private T  height;

        public Dimension( T length, T width, T height) {
                super();
                this.length = length;
                this.width = width;
                this.height = height;
        }
```

Declarando uma Classe utilizando Generics

Classe não trabalha com nenhuma referência a um tipo específico

```
public class ManipulaArray<T> { // Contem o parâmetro <T>

    private T[] array; // atributo de tipo generic

    public ManipulaArray( T[] array ) { // Construtor
        this.array = array;
    }

    public boolean existeElemento(T elementoABuscar)
        for (T elemento : array) {
            if (elemento.equals(elementoABuscar)) {
                return true;
            }
        }
        return false;
    }

    // get e set de atributo generico
    public T[] getArray() { return array;}
    public void setArray(T[] array) {this.array = array;}
}
```

Indica que a classe declarada é uma classe **Generics**

Declarando Métodos Genéricos

Letras mais usadas

The most commonly used type parameter names are:

E - Element (used extensively by the Java Collections Framework)
K - Key
N - Number
T - Type
V - Value
S,U,V etc. - 2nd, 3rd, 4th types

Generics and Primitive Data Types

Uma restrição de genéricos em Java é que o parâmetro type não pode ser um tipo primitivo.

Por exemplo, o seguinte não é compilado:


```
List<int> list = new ArrayList<>();
list.add(17);
```

Para entender por que os tipos de dados primitivos não funcionam, lembre-se de que os genéricos são um recurso em tempo de compilação, o que significa que o parâmetro type é apagado e todos os tipos genéricos são implementados como o Object.

Generics and Primitive Data Types

```
List<Integer> list = new ArrayList<>();
list.add(17);
```

A assinatura do método add é:

```
boolean add(E e);
```

E será compilado para:

```
boolean add(Object e);
```

Generics and Primitive Data Types

Portanto, os parâmetros de tipo devem ser conversíveis em Objeto. Como os tipos primitivos não estendem Object, não podemos usá-los como parâmetros de tipo. No entanto, o Java fornece tipos de boxes para primitivas, juntamente com autoboxing e unboxing para (unwrap) desembrulhá-las:

```
Integer a = 17;
int b = a;
```

Portanto, se queremos criar uma lista que possa conter números inteiros, podemos usar o wrapper:

```
List<Integer> list = new ArrayList<>();
list.add(17);
int first = list.get(0);
```

O código compilado será o equivalente a:

```
List list = new ArrayList<>();
list.add(Integer.valueOf(17));
int first = ((Integer) list.get(0)).intValue();
```

Generics and Primitive Data Types

Versões futuras do Java podem permitir tipos de dados primitivos para genéricos.

O Projeto Valhalla visa melhorar a maneira como os genéricos são tratados.

A idéia é implementar a especialização em genéricos, conforme descrito no JEP 218.

Limitando Genéricos

```
public class ColecaoBichoFelino<T extends Felino> {

    T[] animais;
}
```

Leão

Gato

Coringa <?>

Aceita T e todos os seus ascendentes

< ? super T >

< ? extends T >

Aceita T e todos os seus descendentes

```
public class ColecaoBichoFelino {

    public void addAnimal(List<? extends Felino> animais) {

        //animais.add(new Leao()); //não pode adicionar quando e utilizado
                                   //<? extends Felino>
        for (Felino bicho : animais) {
            bicho.fazerRuido();
        }
    }

    public static void main(String[] args) {

        List<Leao> animais = new ArrayList<Leao>();
        animais.add(new Leao());
        ColecaoBichoFelino colecao = new ColecaoBichoFelino();
        colecao.addAnimal(animais);
    }
}
```

Aceita somente Felino
Neste caso [Leão]

Wildcards Generics

Existem 3 tipos de Wildcards em Generics:

List<?>

mas conhecido como Unknown Wildcard. Wildcard desconhecido.

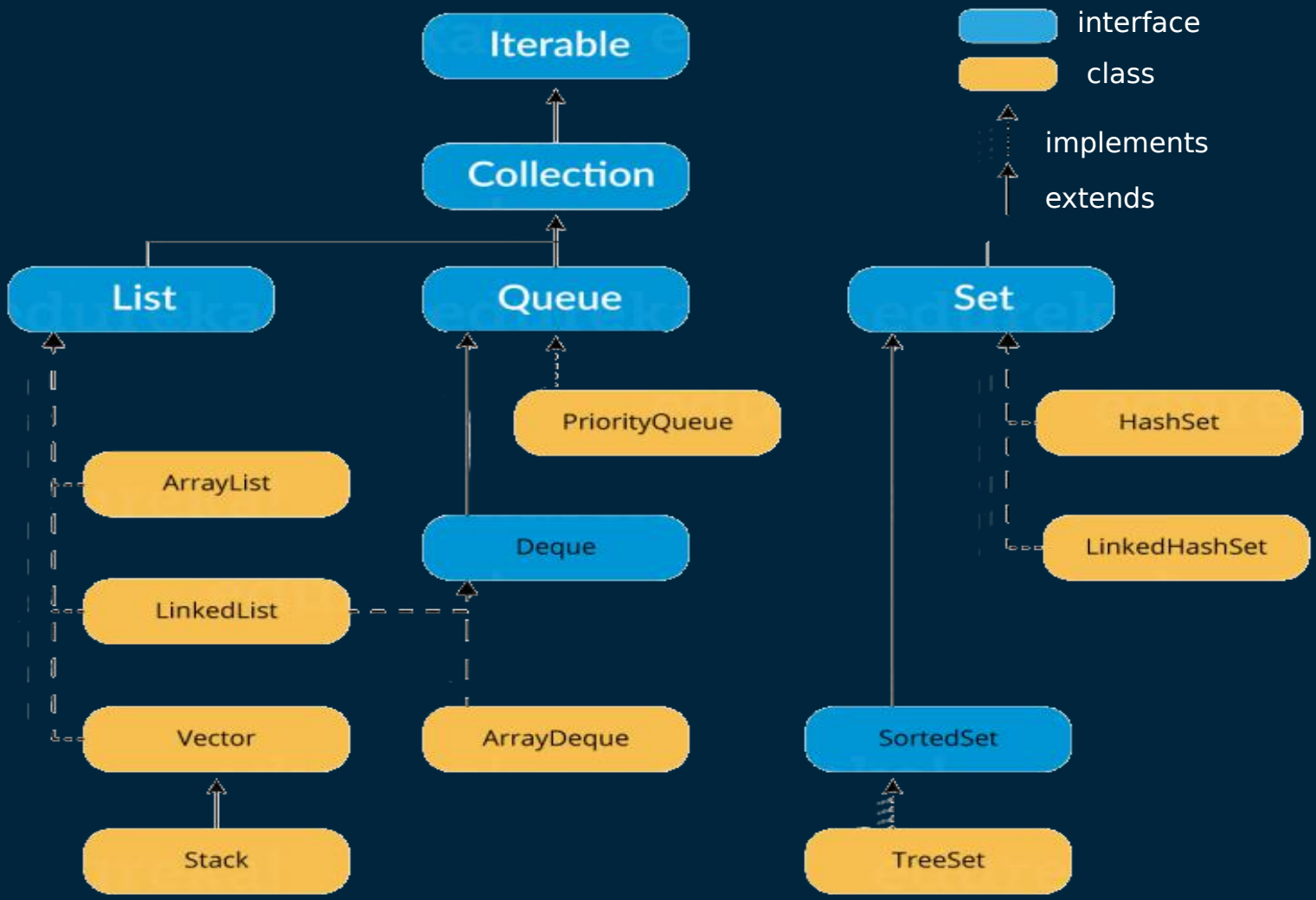
List<? extends A>

List<? super A>

SubTipos

Integer	é um subtipo de Number
Double	é um subtipo de Number
ArrayList<E>	é um subtipo de List<E>
List<E>	é um subtipo de Collection<E>
Collection<E>	é um subtipo de Iterable<E>

Generics in Java



Atividades

```
public static void main(String[] args) {
    // Sintaxe  para a invocação de método genérico
    Pair<Integer, String> p1 = new Pair<>(1, "apple");
    Pair<Integer, String> p2 = new  Pair<>(1, "apple");

    boolean same1 = Util.<Integer, String>compare(p1, p2);
    System.out.println("Compare : "+ same1);

    //Usando inferência  de tipos:
    Pair<String, String> p3 = new Pair<>("AP", "apple");
    Pair<String, String> p4 = new Pair<>("PE", "pear");
    boolean same2 = Util.compare(p3, p4);
    System.out.println("Compare : "+ same2);
}
```

Atividades

```java
public class Pair<K, V> {
    private K key;
    private V value;

    // Generic constructor
    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    // Generic methods
    public void setKey(K key) { this.key = key; }
    public void setValue(V value) { this.value = value; }
    public K getKey() { return key; }
    public V getValue() { return value; }

}
```

Atividades

```java
public class Util {
    // Generic static method
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}
```

Objetivo da atividade criar o projeto acima, criar classe executora criar a classe Generica Pair e uma classe Util e chamar o metodo compare da mesma.
Ao finalizar a mesma, a atividade consiste em criar agora um Pair com três atributos:
firstName: T
SecondName: U
Age: S

VALEU GALERA