Curso
Programação Orientada a Objetos com Java

Capítulo: Generics, Set, Map

-

Introdução aos Generics

-

Generics

- Generics permitem que classes, interfaces e métodos possam ser parametrizados por tipo. Seus benefícios são:
  - Reuso
  - Type safety
  - Performance

- Uso comum: coleções

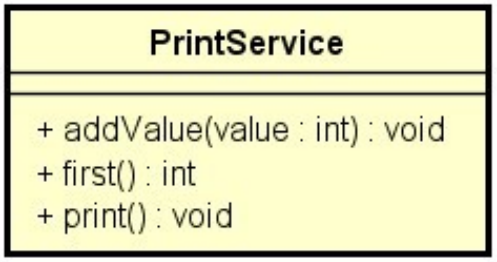
```
List<String> list = new ArrayList<>();
list.add("Maria");
String name = list.get(0);
```

Problema motivador 1 (reuso)

Deseja-se fazer um programa que leia uma quantidade N, e depois N números inteiros. Ao final, imprima esses números de forma organizada conforme exemplo. Em seguida, informar qual foi o primeiro valor informado.

```
How many values? 3
10
8
23
[10,  8,  23]
First: 10
```

Criar um serviço de impressão:

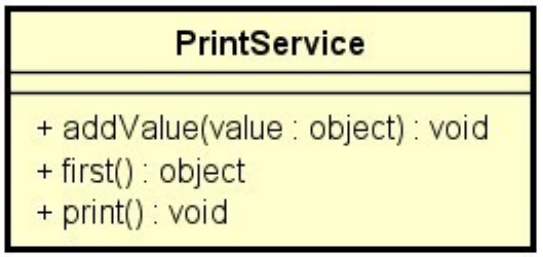


Problema motivador 2 (type safety & performance)

Deseja-se fazer um programa que leia uma quantidade N, e depois N nomes de pessoas. Ao final, imprima esses números de forma organizada conforme exemplo. Em seguida, informar qual foi o primeiro valor informado.

```
How many values? 3
10
8
23
[10,  8,  23]
First: 10
```

Criar um serviço de impressão:

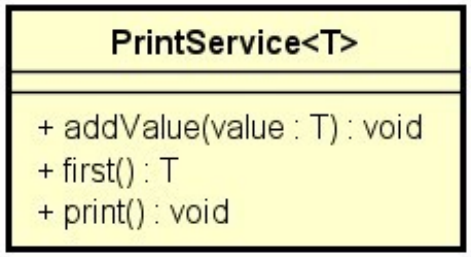


Solução com generics

Deseja-se fazer um programa que leia uma quantidade N, e depois N números inteiros. Ao final, imprima esses números de forma organizada conforme exemplo. Em seguida, informar qual foi o primeiro valor informado.

```
How many values? 3
10
8
23
[10,  8,  23]
First: 10
```

Criar um serviço de impressão:



Genéricos delimitados

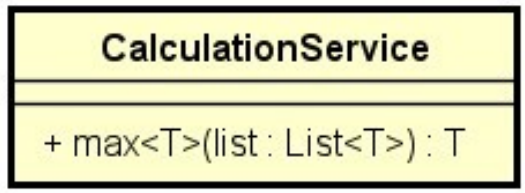
-

Problema

Uma empresa de consultoria deseja avaliar a performance de produtos, funcionários, dentre outras coisas. Um dos cálculos que ela precisa é encontrar o maior dentre um conjunto de elementos. Fazer um programa que leia um conjunto de produtos a partir de um arquivo, conforme exemplo, e depois mostre o mais caro deles.

```
Computer,890.50
IPhone X,910.00
Tablet,550.00
Most expensive:
IPhone, 910.00
```

Criar um serviço de cálculo:

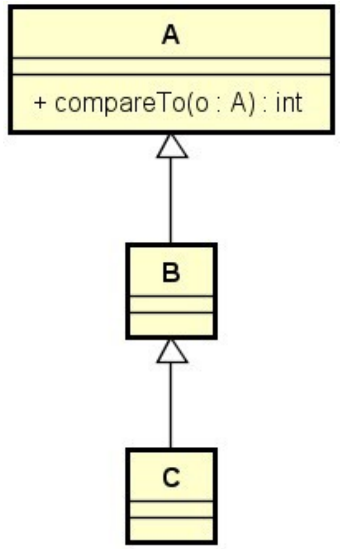


Nota: Java possui:
Collections.max(list)

```java
package services;

import java.util.List;

public class CalculationService {

    public static <T extends Comparable<T>> T max(List<T> list) {
        if (list.isEmpty()) {
            throw new IllegalStateException("List can't be empty");
        }
        T max = list.get(0);
        for (T item : list) {
            if (item.compareTo(max) > 0) {
                max = item;
            }
        }
        return max;
    }
}
```

Versão alternativa (completa)

```
public static <T extends Comparable<? super T>> T max(List<T> list) {
    if (list.isEmpty()) {
        throw new IllegalStateException("List can't be empty");
    }
    T max = list.get(0);
    for (T item : list) {
        if (item.compareTo(max) > 0) {
            max = item;
        }
    }
    return max;
}
```



Tipos curinga (wildcard types)

-

Generics são invariantes

List<Object> não é o supertipo de qualquer tipo de lista:

```
List<Object> myObjs = new ArrayList<Object>();
List<Integer>  myNumbers  =  new  ArrayList<Integer>();
myObjs = myNumbers; // erro de compilação
```

O supertipo de qualquer tipo de lista é List<?>. Este é um tipo curinga:

```
List<?> myObjs = new ArrayList<Object>();
List<Integer>  myNumbers  =  new  ArrayList<Integer>();
myObjs = myNumbers;
```

Com tipos curinga podemos fazer métodos que recebem um genérico de "qualquer tipo":

```
package application;

import java.util.Arrays;
import java.util.List;

public class Program {

    public static void main(String[] args) {
        List<Integer> myInts = Arrays.asList(5, 2, 10);
        printList(myInts);
    }

    public static void printList(List<?> list) {
        for (Object obj : list) {
            System.out.println(obj);
        }
    }
}
```

Porém não é possível adicionar dados a uma coleção de tipo curinga

```
package application;

import   java.util.ArrayList;
import java.util.List;

public class Program {

    public static void main(String[] args) {

        List<?> list = new ArrayList<Integer>();
        list.add(3); // erro de compilação
    }
}
```

O compilador não sabe qual é o tipo específico do qual a lista foi instanciada.

Curingas delimitados (bounded wildcards)

-

Problema 1

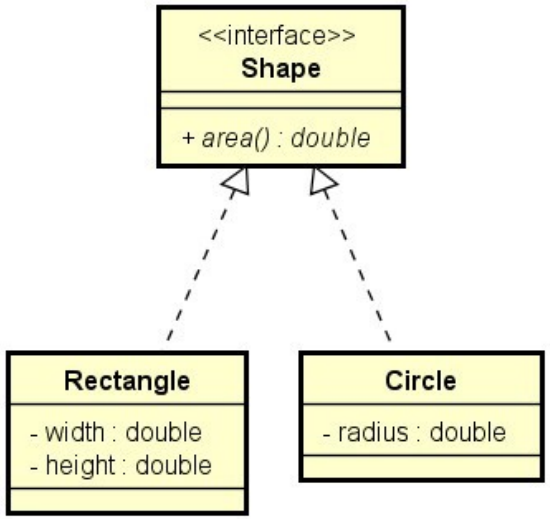
Vamos fazer um método para retornar a soma das áreas de uma lista de figuras.

Nota 1: soluções impróprias:

```
public  double  totalArea(List<Shape>  list)
```

```
public double totalArea(List<?> list)
```

Nota 2: não conseguiremos adicionar elementos na lista do método



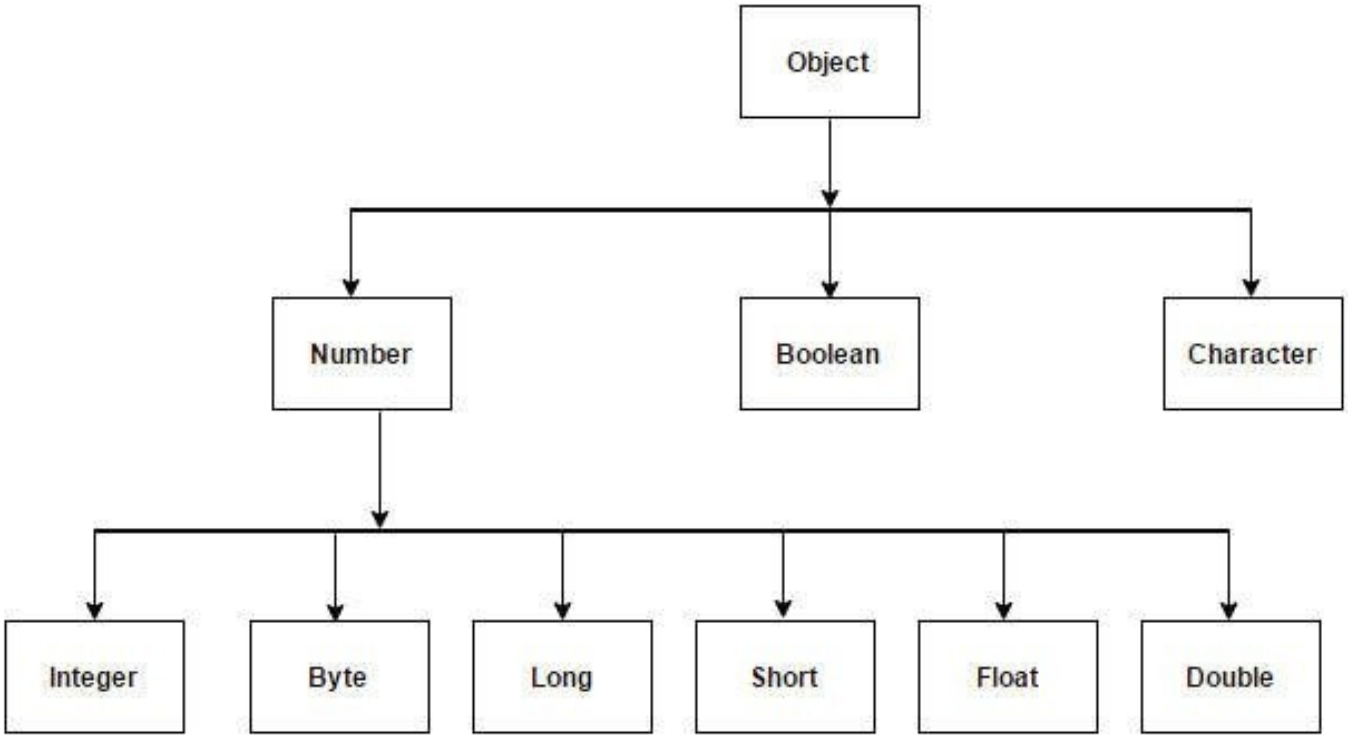
Problema 2 (princípio *get/put*)

Vamos fazer um método que copia os elementos de uma lista para uma outra lista que pode ser mais genérica que a primeira.

```
List<Integer> myInts = Arrays.asList(1, 2, 3, 4);
List<Double> myDoubles = Arrays.asList(3.14, 6.28);
List<Object> myObjs = new ArrayList<Object>();

copy(myInts, myObjs);

copy(myDoubles, myObjs);
```

https://stackoverflow.com/questions/1368166/what-is-a-difference-between-super-e-and-extends-e

Java wrapper types (próximos exemplos)



Princípio get/put - covariância

```
List<Integer> intList = new ArrayList<Integer>();
intList.add(10);
intList.add(5);

List<? extends Number> list = intList;

Number x = list.get(0);

list.add(20); // erro de compilacao
```

get - OK

put - ERROR

Princípio get/put - contravariância

```
List<Object>  myObjs  =  new  ArrayList<Object>();
myObjs.add("Maria");
myObjs.add("Alex");

List<? super Number> myNums = myObjs;

myNums.add(10);
myNums.add(3.14);

Number x = myNums.get(0); // erro de compilacao
```

get - ERROR

put - OK

```java
package application;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Program {

	public static void main(String[] args) {

		List<Integer> myInts = Arrays.asList(1, 2, 3, 4);
		List<Double> myDoubles = Arrays.asList(3.14, 6.28);
		List<Object> myObjs = new ArrayList<Object>();

		copy(myInts, myObjs);
		printList(myObjs);
		copy(myDoubles, myObjs);
		printList(myObjs);
	}

	public static void copy(List<? extends Number> source, List<? super Number> destiny) {
	    for(Number number : source) {
	        destiny.add(number);
	    }
	}

	public static void printList(List<?> list) {
		for (Object obj : list) {
			System.out.print(obj + " ");
		}
		System.out.println();
	}
}
```

hashCode e equals

hashCode e equals

• São operações da classe Object utilizadas para comparar se um objeto é igual a outro

• equals: lento, resposta 100%

• hashCode: rápido, porém resposta positiva não é 100%

• Tipos comuns (String, Date, Integer, Double, etc.) já possuem implementação para essas operações. Classes personalizadas precisam sobrepô-las.

Equals

Método que compara se o objeto é igual a outro, retornando true ou false.

```
String a = "Maria";
String b = "Alex";

System.out.println(a.equals(b));
```

HashCode

Método que retorna um número inteiro representando um código gerado a partir das informações do objeto

```
String a = "Maria";
String b = "Alex";

System.out.println(a.hashCode());
System.out.println(b.hashCode());
```

Regra de ouro do HashCode

• Se o hashCode de dois objetos for diferente, então os dois objetos são diferentes

Isso nunca acontece:

"Alex Larry Brown"	-242670543
"Alex Larry Brown"	880483901

• Se o código de dois objetos for igual, muito provavelmente os objetos são iguais (pode haver colisão)

HashCode e Equals personalizados

```
public class Client {

    private String name;
    private String email;
}
```

Set

-

Set<T>

- Representa um conjunto de elementos (similar ao da Álgebra)
  - Não admite repetições
  - Elementos não possuem posição
  - Acesso, inserção e remoção de elementos são rápidos
  - Oferece operações eficientes de conjunto: interseção, união, diferença.
  - Principais implementações:
    - HashSet - mais rápido (operações O(1) em tabela hash) e não ordenado
    - TreeSet - mais lento (operações O(log(n)) em árvore rubro-negra) e ordenado pelo compareTo do objeto (ou Comparator)
    - LinkedHashSet - velocidade intermediária e elementos na ordem em que são adicionados

- https://docs.oracle.com/javase/10/docs/api/java/util/Set.html

Alguns métodos importantes

- add(obj), remove(obj), contains(obj)
  - Baseado em equals e hashCode
  - Se equals e hashCode não existir, é usada comparação de ponteiros

- clear()
- size()
- removeIf(predicate)

- addAll(other) - união: adiciona no conjunto os elementos do outro conjunto, sem repetição
- retainAll(other) - interseção: remove do conjunto os elementos não contitos em other
- removeAll(other) - diferença: remove do conjunto os elementos contidos em other

Demo 1

```
package  application;

import   java.util.HashSet;
import java.util.Set;

import  Entities.Product;

public class  Program {

     public static void  main(String[] args) {

          Set<String> set  = new HashSet<>();

          set.add("TV");
          set.add("Notebook");
          set.add("Tablet");

          System.out.println(set.contains("Notebook"));

          for (String  p : set) {
               System.out.println(p);
          }
     }
}
```

Demo 2

```
package application;

import java.util.Arrays;
import java.util.Set;
import java.util.TreeSet;

public class Program {

	public static void main(String[] args) {

		Set<Integer> a = new TreeSet<>(Arrays.asList(0,2,4,5,6,8,10));
		Set<Integer> b = new TreeSet<>(Arrays.asList(5,6,7,8,9,10));

		//union
		Set<Integer> c = new TreeSet<>(a);
		c.addAll(b);
		System.out.println(c);

		//intersection
		Set<Integer> d = new TreeSet<>(a);
		d.retainAll(b);
		System.out.println(d);

		//difference
		Set<Integer> e = new TreeSet<>(a);
		e.removeAll(b);
		System.out.println(e);
	}
}
```

Como Set testa igualdade?

-

Como as coleções Hash testam igualdade?

- Se hashCode e equals estiverem implementados:
  - Primeiro hashCode. Se der igual, usa equals para confirmar.
  - Lembre-se: String, Integer, Double, etc. já possuem equals e hashCode
- Se hashCode e equals NÃO estiverem implementados:
  - Compara as referências (ponteiros) dos objetos.

Demo

```
package  application;

import   java.util.HashSet;
import java.util.Set;

import  Entities.Product;

public class  Program {

     public static void  main(String[] args) {

          Set<Product> set = new HashSet<>();

          set.add(new  Product("TV", 900.0));
          set.add(new   Product("Notebook", 1200.0));
          set.add(new  Product("Tablet", 400.0));

          Product prod  = new  Product("Notebook", 1200.0);

          System.out.println(set.contains(prod));
     }
}
```

```java
package  entities;

public class  Product {

     private   String   name;
     private  Double price;

     public Product(String name, Double price) {
          this.name = name;
          this.price = price;
     }

     public String getName() {
          return name;
     }

     public void  setName(String  name) {
          this.name = name;
     }

     public Double getPrice() {
          return price;
     }

     public void  setPrice(Double price) {
          this.price = price;
     }
}
```

Como TreeSet compara os elementos?

-

Recordando as implementações

- Principais implementações:
  - HashSet - mais rápido (operações O(1) em tabela hash) e não ordenado
  - TreeSet - mais lento (operações O(log(n)) em árvore rubro-negra) e ordenado pelo compareTo do objeto (ou Comparator)
  - LinkedHashSet - velocidade intermediária e elementos na ordem em que são adicionados

Demo

```
package  application;

import java.util.Set;
import java.util.TreeSet;

import  Entities.Product;

public class  Program {

     public static void  main(String[] args) {

          Set<Product> set = new TreeSet<>();

          set.add(new  Product("TV", 900.0));
          set.add(new  Product("Notebook", 1200.0));
          set.add(new  Product("Tablet", 400.0));

          for (Product  p  : set) {
               System.out.println(p);
          }
     }
}
```

```java
package entities;

public class Product implements Comparable<Product> {

    private String name;
    private Double price;

    public Product(String name, Double price) {
        this.name = name;
        this.price = price;
    }

    // (... get / set / hashCode / equals)

    @Override
    public String toString() {
        return "Product [name=" + name + ", price=" + price + "]";
    }

    @Override
    public int compareTo(Product other) {
        return  name.toUpperCase().compareTo(other.getName().toUpperCase());
    }
}
```

Exercício resolvido (Set)

-

Problema exemplo

Um site de internet registra um log de acessos dos usuários. Um registro de log consiste no nome de usuário (apenas uma palavra) e o instante em que o usuário acessou o site no padrão ISO 8601, separados por espaço, conforme exemplo. Fazer um programa que leia o log de acessos a partir de um arquivo, e daí informe quantos usuários distintos acessaram o site.

Example

input file:

```
amanda 2018-08-26T20:45:08Z
alex86 2018-08-26T21:49:37Z
bobbrown 2018-08-27T03:19:13Z
amanda 2018-08-27T08:11:00Z
jeniffer3 2018-08-27T09:19:24Z
alex86 2018-08-27T22:39:52Z
amanda 2018-08-28T07:42:19Z
```

Execution:

```
Enter file full path: c:\temp\in.txt
Total users: 4
```

Exercício proposto (Set)

-

Em um portal de cursos online, cada usuário possui um código único, representado por um número inteiro.

Cada instrutor do portal pode ter vários cursos, sendo que um mesmo aluno pode se matricular em quantos cursos quiser. Assim, o número total de alunos de um instrutor não é simplesmente a soma dos alunos de todos os cursos que ele possui, pois pode haver alunos repetidos em mais de um curso.

O instrutor Alex possui três cursos A, B e C, e deseja saber seu número total de alunos.

Seu programa deve ler os alunos dos cursos A, B e C do instrutor Alex, depois mostrar a quantidade total e alunos dele, conforme exemplo.

Example:

How many students for course A? 3
21
35
22
How many students for course B? 2
21
50
How many students for course C? 3
42
35
13
Total students: 6

Map

-

Map<K,V>

- https://docs.oracle.com/javase/10/docs/api/java/util/Map.html

- É uma coleção de pares chave / valor
  - Não admite repetições do objeto chave
  - Os elementos são indexados pelo objeto chave (não possuem posição)
  - Acesso, inserção e remoção de elementos são rápidos

- Uso comum: cookies, local storage, qualquer modelo chave-valor

- Principais implementações:
  - HashMap - mais rápido (operações O(1) em tabela hash) e não ordenado
  - TreeMap - mais lento (operações O(log(n)) em árvore rubro-negra) e ordenado pelo compareTo do objeto (ou Comparator)
  - LinkedHashMap - velocidade intermediária e elementos na ordem em que são adicionados

Alguns métodos importantes

- put(key, value), remove(key), containsKey(key), get(key)
  - Baseado em equals e hashCode
  - Se equals e hashCode não existir, é usada comparação de ponteiros

- clear()
- size()

- keySet() - retorna um Set<K>
- values() - retornaa um Collection<V>

Demo 1

```
package application;

import java.util.Map;
import java.util.TreeMap;

public class Program {

	public static void main(String[] args) {

		Map<String, String> cookies = new TreeMap<>();

		cookies.put("username", "maria");
		cookies.put("email",   "maria@gmail.com");
		cookies.put("phone", "99771122");

		cookies.remove("email");
		cookies.put("phone", "99771133");

		System.out.println("Contains 'phone' key: " + cookies.containsKey("phone"));
		System.out.println("Phone number: " + cookies.get("phone"));
		System.out.println("Email: " + cookies.get("email"));
		System.out.println("Size: " + cookies.size());

		System.out.println("ALL COOKIES:");
		for (String key : cookies.keySet()) {
			System.out.println(key + ": " + cookies.get(key));
		}
	}
}
```

Demo 2

```
package  application;

import   java.util.HashMap;
import java.util.Map;

import  Entities.Product;

public class  Program {

     public static void  main(String[] args) {

          Map<Product, Double> stock  = new HashMap<>();

          Product p1 = new Product("Tv", 900.0);
          Product p2 = new Product("Notebook", 1200.0);
          Product p3 = new Product("Tablet", 400.0);

          stock.put(p1, 10000.0);
          stock.put(p2, 20000.0);
          stock.put(p3, 15000.0);

          Product ps = new Product("Tv", 900.0);

          System.out.println("Contains 'ps' key: " + stock.containsKey(ps));
     }
}
```

```java
package entities;

public class Product {

    private  String  name;
    private Double price;

    public  Product(String name, Double price) {
        this.name = name;
        this.price  =  price;
    }

    // getters, setters, equals, hashCode
}
```

Exercício proposto (Map)

-

Na contagem de votos de uma eleição, são gerados vários registros de votação contendo o nome do candidato e a quantidade de votos (formato .csv) que ele obteve em uma urna de votação. Você deve fazer um programa para ler os registros de votação a partir de um arquivo, e daí gerar um relatório consolidado com os totais de cada candidato.

Input file example:

```
Alex Blue,15
Maria Green,22
Bob Brown,21
Alex Blue,30
Bob Brown,15
Maria Green,27
Maria Green,22
Bob Brown,25
Alex Blue,31
```

Execution:

```
Enter file full path:  c:\temp\in.txt
Alex Blue: 76
Maria Green: 71
Bob Brown: 61
```

Solução do exercício