Ferramentas Java para estruturas de dados

Prof. Tiago Massoni

Engenharia da Computação

Poli - UPE

Herança em Java

```
public class Poupanca extends Conta {
  //atributos especificos de poupanca
  //metodos especificos de poupanca
}
```

- Restrição de Java
  - Uma classe pode estender apenas uma superclasse diretamente

Herança

- Estrutura herdada
  - Atributos, métodos
- Construtores
  - **não são herdados -> precisam ser implementados na subclasse**
  - construtores da subclasse "**sempre**" utilizam algum construtor da superclasse.

Construtores e subclasses

```
public class Poupanca extends Conta {
  public Poupanca (String num,double saldo,
                   Cliente cliente){
     super(num, saldo, cliente);
  }
}
```

super chama o construtor da superclasse

se **super** não for chamado, o compilador acrescenta uma chamada ao construtor default: **super()**

⇨

se não existir um construtor default na superclasse, haverá um erro de compilação

Interface

- Um "supertipo" definindo serviços em comum
- Nenhuma herança envolvida

```
public interface QualquerBanco {
    public double saldoTotal();
    public int numElementos();
}
```


Interfaces

- Todos os métodos são abstratos
  - Não têm implementação!
  - provêem uma interface para serviços
  - são qualificados como **public** automaticamente
- não definem atributos
  - apenas podem definir constantes
  - por default todos os "atributos" definidos em uma interface são qualificados como **public**, **static** e **final**
- não definem construtores

Subtipos

obrigada "por contrato" a implementar todos os métodos da interface

```
public class BancoSeguros implements
                        QualquerBanco {
 public double saldoTotal(){
     //calcula soma das apolices
 }
 public int numElementos(){
     //calcula numero de apólices
 }
 /*...outros metodos...*/
}

public class BancoInvest implements
                        QualquerBanco {
    /* ...mesma coisa, com poupancas... */
}
```

Entrada e saída

- Pacote java.io
- Uso de streams (fluxo sequencial)
  - Entrada padrão: System.in
  - Saída padrão: System.out
- Classe Console (java 5)
  - Possui operações de uso da entrada e da saída padrão

Console

```
Console c = System.console();
if (c == null) {
   System.err.println("No console.");
   System.exit(1);
}
String login = c.readLine("Enter your login:");
char[]oldPassword = c.readPassword("Password: ");
```

- Outros métodos
  - readLine (format,string)
  - printf(format,string)

API Scanner

- Classe java.util.Scanner possibilita leitura de tokens separados por algum caracter (default: espaço)
  - Tokens com algum tipo específico

```
Scanner sc = new Scanner(System.in);
while (sc.hasNextInt())
      int i = sc.nextInt();

Scanner sc = new Scanner(new File("myNumbers"));
while (sc.hasNextLong()) {
      long aLong = sc.nextLong();
}
```


Tratamento de exceções

```
public class Conta {
  private String numero;
  private double saldo;
  ...
  public void debitar(double valor) {
    saldo = saldo - valor;
  }
}
```

Como evitar débitos acima do limite permitido?

Solução: exceções em Java

Define o construtor com mensagem

Todas as exceções estendem java.lang.Exception

```
public class SaldoNegativoException extends
                        Exception {
  public SaldoNegativoException(String num) {
    super ("Saldo Insuficiente na conta:"+num);
  }
}
```

Passo 1: definir a sua exceção

Exceções

Anuncia "eu posso lançar essa daqui!!"

```
public class Conta {
  ...
  public void debitar(double valor)
                throws SaldoNegativoException {
    if (valor <= saldo)
     saldo = saldo - valor;
    else {
     throw new SaldoNegativoException(this.numero);
    }
  }
}
```

Ops, aconteceu! Criar objeto e lançar

Passo 2: lançar exceção ao ocorrer o erro

Exceções

Bloco try/catch agarra as possíveis exceções anunciadas

```
public class Programa {
  ...
  public static void main (String[] args){
    Conta c = new Conta(..,0.0);
    try {
     c.debitar(10.0);
    } catch (SaldoNegativoException e) {
     System.out.println(e.getMessage());
    }
}
```

Mensagem que veio com a exceção

Passo 3: Tratamento das exceções

Tratando exceções: forma geral

```
try {
  ...
} catch (E1 e1) {
  ...
}
...
} catch (En en) {
  ...
} finally {
  ...
}
```

O bloco finally é sempre executado, qualquer seja o resultado

API collection

- Framework com interfaces e implementações padrão para certas estruturas de dados e algoritmos que as manipulam
- Não as usaremos diretamente na maioria dos casos
- Classes utilitárias
  - java.util.Arrays
  - java.util.Collections

java.util.Arrays

- copyOf(tipo[] original, novoTam)
- copyOfRange(tipo[] original,from,to)
- equals(tipo[] a1,tipo[] a2)
- fill(tipo[] a, tipo valor)
- toString(tipo [] a)

Generics (java 1.5)

- Problema: métodos e classes que funcionam para um tipo específico
  - Repetição de código
- Solução: nível mais alto de generalidade
  - Parametrização de classes e métodos por tipo

Problema

Lista aceita Object

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

Cast (checagem em tempo de execução)

```
List<Integer> myIntList= new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

Compilador não aceitaria objeto que não Integer!

Exemplos generics

```
public interface List <E>{
    void add(E x);
    Iterator<E> iterator();
}

public interface Iterator <E>{
    E next();
    boolean hasNext();
}
```


Métodos genéricos

```
static <T> void
     fromArrayToCollection(T[] a, Collection<T> c){
    for (T o : a) {
        c.add(o);
    }
}
```


Interface java.lang.Comparable

public class Estudante implements Comparable

- Deve implementar método
  public int compareTo(Object o)
  - Parâmetro é Object
  - Deve retornar
    - Inteiro negativo = menor que
    - Zero = iguais
    - Inteiro positivo = maior que

```
public int compareTo(Object o) {
  if (o instanceof Estudante)
    return pontos - ((Estudante)o).getPontos();
}
```


Comparator

- Comparator é uma classe utilitária (java.util)
- Use quando objetos não forem Comparable ou quando não quiser usar critério de ordenação original do objeto
- Método :
  - compare() compara dois objetos recebidos

Exemplo

```
import java.util.*;
public class DataUtil implements Comparator{
  ...
  public int compare(Object obj1, Object obj2){
      Data dt1 = (Data) obj1;
      Data dt2 = (Data) obj2;
      int retorno = dt1.compareTo(dt2);
      if (retorno!=0) retorno *=-1;
        return retorno;
  }
}
```