

Programação Orientada a Objetos (PO24CP-4CP)

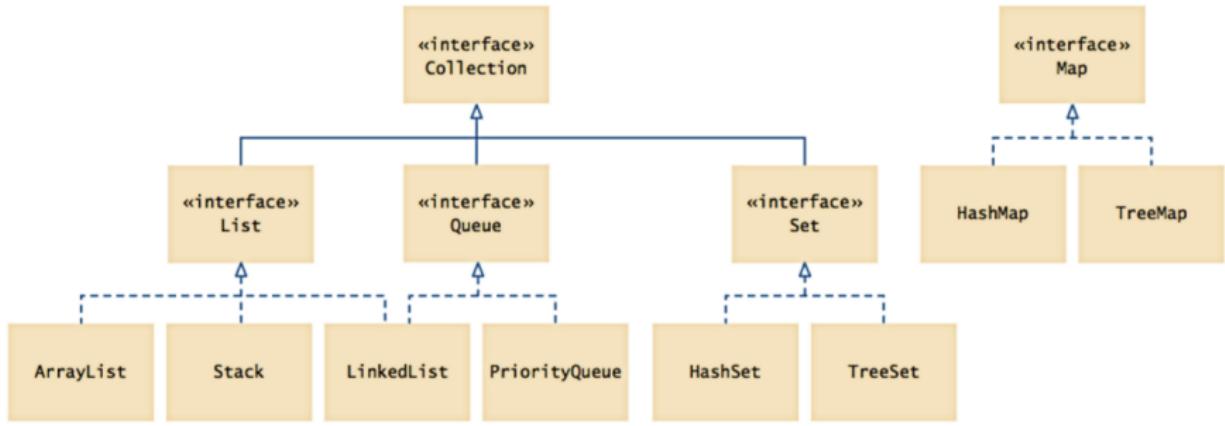
Aula #13 - Coleções

Prof^a Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Coleções

Coleções

- Java provê uma API (*Application Programming Interface*) para armazenamento de dados (instâncias de classes) na memória
 - Contém classes e interfaces com mecanismos para agrupar e processar objetos em conjuntos.
 - Simples e flexível.
 - Com diversas modalidades e variantes.
 - Equivalem às principais estruturas clássicas de dados.
- Coleções: API de estruturas de dados.



- Outras características interessantes para manipulação de coleções:
 - Laço for para iteração.
 - *Autoboxing*: conversão automática entre um tipo primitivo e seu respectivo objeto de classe *wrapper* (empacotadora)
- Classe Collections tem métodos estáticos úteis.
 - Ex.: método estático `sort`

Estruturas de Dados: Sets

- Coleção de objetos que não admite objetos em duplicata.
- Interface Set: define contrato.
- Métodos:
 - add: adiciona um objeto ao set.
 - remove: remove um objeto do set.
 - contains: retorna true se o set contém o objeto.
- Classe HashSet: set com boa performance.
- Classe TreeSet: set que garante que elementos estarão em ordem implícita.

Estruturas de Dados: Sets

```
import java.util.Date;
import java.util.HashSet;

public class ExemploSet1
{ public static void main(String[] args)
{
    HashSet<Object> set = new HashSet<Object>();
    set.add(new Integer(123));
    set.add(123);
    set.add("ABC");
    set.add("ABC");
    set.add(new Date());
    set.remove("ABC");
    System.out.println(set);
    // [Wed Nov 09 13:57:51 BRST 2016, 123]
}
```

Estruturas de Dados: Sets

```
import java.util.Date;
import java.util.TreeSet;

public class ExemploSet2
{
    public static void main(String[] args)
    {
        TreeSet<Object> set = new TreeSet<Object>();
        set.add(new Integer(123));
        set.add(123);
        set.add("ABC"); // !!!!!
        set.add("ABC");
        set.add(new Date());
        set.remove("ABC");
        System.out.println(set);
    }
}
```

Exception in thread "main" java.lang.ClassCastException:
java.lang.Integer cannot be cast to java.lang.String

Estruturas de Dados: Sets

```
import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;

public class ExemploSet3
{
    public static void main(String[] args)
    {
        HashSet<Object> set = new HashSet<Object>();
        set.add(new Integer(123)); set.add(123);
        set.add("ABC"); set.add("ABC");
        set.add(new Date());
        Iterator<Object> i = set.iterator();
        while(i.hasNext())
        {
            Object o = i.next();
            if (o instanceof Integer)
                System.out.println("Achei um Integer:" + o);
        }
        // Achei um Integer:123
    }
}
```

Estruturas de Dados: Sets

```
import java.util.HashSet;

public class ExemploSet4
{
    public static void main(String[] args)
    {
        HashSet<Integer> set = new HashSet<Integer>();
        set.add(new Integer(123));
        set.add(111);
        set.add(Integer.parseInt("877"));
        // set.add("877"); Erro de compilação: método não aplicável
        set.add(123);
        int sum = 0;
        System.out.print("Soma de ");
        for (Integer i : set)
        {
            System.out.print(i + " ");
            sum += i;
        }
        System.out.println("é " + sum);
        // Soma de 111 877 123 é 1111]
    }
}
```

- Mais operações úteis em *Sets*:
 - addAll: adiciona um *set* a outro.
 - retainAll: retém em um *set* tudo o que estiver em outro: interseção de sets.
 - removeAll: remove de um *set* tudo o que estiver em outro.
 - containsAll: retorna true se o *set* conter todos os elementos de outro.
- Estes métodos recebem como argumento uma outra instância de *Set* e retornam uma nova instância resultante da operação.

Estruturas de Dados: Sets

```
import java.util.HashSet;
public class OperacoesSet{
    public static void main(String[] args){
        HashSet<String> solteiros = new HashSet<String>();
        solteiros.add("Tom");
        solteiros.add("Larry");
        HashSet<String> casados = new HashSet<String>();
        casados.add("Nathan");
        casados.add("Jeffrey");
        casados.add("Randal");
        casados.add("Sriram");
        HashSet<String> tenistas = new HashSet<String>();
        tenistas.add("Tom");
        tenistas.add("Jeffrey");
        tenistas.add("Larry");
        HashSet<String> nadadores = new HashSet<String>();
        nadadores.add("Nathan");
        nadadores.add("Sriram");
        nadadores.add("Tom");
        // Todos os autores
        HashSet<String> todos = new HashSet<String>(casados);
        todos.addAll(solteiros);
        // [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]
```

Estruturas de Dados: Sets

```
// Nadadores e tenistas
HashSet<String> nadadoresETenistas = new HashSet<String>(nadadores);
nadadoresETenistas.retainAll(tenistas); // [Tom]
// Tenistas e casados
HashSet<String> tenistasCasados = new HashSet<String>(tenistas);
tenistasCasados.retainAll(casados);
System.out.println(tenistasCasados); // [Jeffrey]
// Tenistas ou casados
HashSet<String> tenistasOuCasados = new HashSet<String>(tenistas);
tenistasOuCasados.addAll(casados);
System.out.println(tenistasOuCasados);
// [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]
// Casados mas não atletas
HashSet<String> casadosMasNãoAtletas = new HashSet<String>(casados);
casadosMasNãoAtletas.removeAll(tenistas);
casadosMasNãoAtletas.removeAll(nadadores); // [Randal]
// Todo nadador é tenista ?
System.out.println(tenistas.containsAll(nadadores)); // false
// Todo solteiro é tenista ?
System.out.println(tenistas.containsAll(solteiros)); // true
}
}
```

- Coleção de objetos em forma de lista, aceita duplicatas.
- Interface List: define contrato.
- Métodos:
 - add: adiciona um objeto à lista.
 - remove: remove um objeto da lista.
 - get: recupera um objeto da lista.
 - contains: retorna true se a lista contém o objeto.
 - indexOf: retorna o índice do objeto na lista ou -1.
 - size: retorna o número de elementos na lista.

- Classe LinkedList: performance razoável em todas as condições.
- Classe ArrayList: boa performance, mas pode cair quando tamanho é redimensionado.
- Classe Stack: métodos adicionais para push e pop.

Estruturas de Dados: Listas

```
import java.util.ArrayList;
import java.util.Date;
public class ExemploLista1
{
    public static void main(String[] args)
    {
        ArrayList<Object> lista = new ArrayList<Object>();
        lista.add(new Integer(123));
        lista.add(123);
        lista.add("ABC");
        lista.add("ABC");
        lista.add(new Date());
        lista.remove("ABC");
        System.out.println(lista);
        // [123, 123, ABC, Wed Nov 09 14:12:48 BRST 2016]
    }
}
```

Estruturas de Dados: Listas

```
import java.util.LinkedList;
public class ExemploLista2
{
    public static void main(String[] args)
    {
        LinkedList<Float> lista = new LinkedList<Float>();
        lista.add(new Float(1.4));
        lista.add(1f);
        lista.add(new Float(2.61));
        float sum = 0;
        System.out.print("Soma de ");
        for(Float f:lista)
        {
            System.out.print(f+" ");
            sum += f;
        }
        System.out.println(" e' "+sum);
        // Soma de 1.4 1.0 2.61 e' 5.01
    }
}
```

Estruturas de Dados: Listas

```
import java.util.ArrayList;
import java.util.Collections;
public class Sorteio
{
    public static void main(String[] args)
    {
        ArrayList<Integer> números = new ArrayList<Integer>(60);
        for(int i=1;i<60;i++) números.add(i);
        Collections.shuffle(números);
        for(int i=0;i<6;i++) System.out.print(números.get(i)+" ");
        // 7 59 16 51 36 58
    }
}
```

Estruturas de Dados: Mapas

- Também conhecidos como *arrays associativos*.
- Coleção de objetos como *arrays*, mas índices são objetos.
- Outra interpretação: conjunto de pares (chave,valor) de objetos. Chaves não podem ser duplicadas.
- Interface Map: define contrato.
- Métodos:
 - put: adiciona um objeto ao mapa.
 - remove: remove um objeto do mapa.
 - get: recupera um objeto do mapa.
 - keySet: retorna um set com todas as chaves.
 - values: retorna uma coleção com todos os valores.

Estruturas de Dados: Mapas

- Classe HashMap: boa performance.
- Classe TreeMap: elementos ordenados por chave.

```
import java.util.HashMap;
public class ExemploMap1
{
    public static void main(String[] args)
    {
        HashMap<Object, Object> mapa = new HashMap<Object, Object>();
        mapa.put(1, "um");
        mapa.put(2, "dois");
        mapa.put(3, "quatro");
        mapa.put(3, "três");
        //mapa.remove("dois"); // ?
        mapa.remove(2); // ok
        mapa.put(0.0, "zero");
        mapa.put(0, "zero");
        System.out.println(mapa); // {1=um, 3=três, 0=zero, 0.0=zero}
    }
}
```

Estruturas de Dados: Mapas

```
import java.util.TreeMap;
public class ExemploMap2
{
    public static void main(String[] args)
    {
        TreeMap<String, Integer> mapa = new TreeMap<String, Integer>();
        mapa.put("um", 1);
        mapa.put("dois", 2);
        mapa.put("tres", 3);
        mapa.put("quatro", 4);
        mapa.put("cinco", 5);
        System.out.println(mapa);
        // {cinco=5, dois=2, quatro=4, tres=3, um=1}
        System.out.println(mapa.get("quatro") + mapa.get("dois")); // 6
    }
}
```

Exemplo simulação simples: Tanques

Exemplo - simulação simples

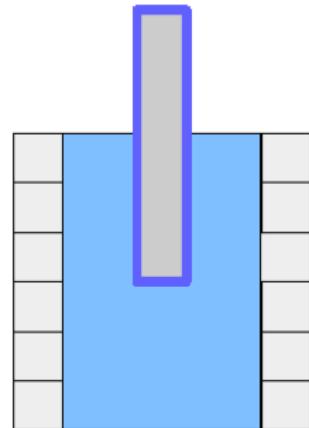
- Primeiros passos em uma simulação muito simples.
- **Tanques** podem andar para a frente, modificar a velocidade, girar nos sentidos horário e anti-horário.
- **Arena** comporta vários tanques e permite a manipulação dos mesmos através do *mouse*.
- **Aplicação** cria instância da Arena.
- **Arena** é um componente bastante específico, Tanque não.

Classe Tanque

```
package tanques;

import java.awt.*;
import java.awt.geom.AffineTransform;

public class Tanque
{
    private double x,y;
    private double ângulo;
    private double velocidade;
    private Color cor;
    private boolean estáAtivo;
    public Tanque(int x,int y,int a,Color c)
    {
        this.x = x; this.y = y; ângulo = 90-a; cor = c;
        velocidade = 0;
        estáAtivo = false;
    }
    // ...
}
```

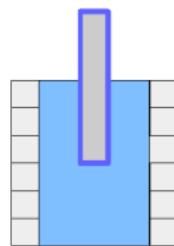


Classe Tanque

```
public void aumentaVelocidade()
{
    velocidade++;
}
public void giraHorário(int a)
{
    ângulo += a;
}
public void giraAntiHorário(int a)
{
    ângulo -= a;
}
public void move()
{
    x = x + Math.sin(Math.toRadians(ângulo))*velocidade;
    y = y - Math.cos(Math.toRadians(ângulo))*velocidade;
}
public void setEstáAtivo(boolean estáAtivo)
{
    this.estáAtivo = estáAtivo;
}
```

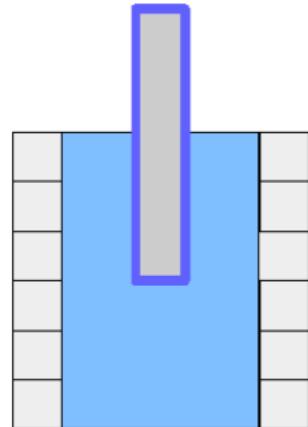
Classe Tanque

```
public void draw(Graphics2D g2d)
{
    // Armazenamos o sistema de coordenadas original.
    AffineTransform antes = g2d.getTransform();
    // Criamos um sistema de coordenadas para o robô.
    AffineTransform at = new AffineTransform();
    at.translate(x,y);
    at.rotate(Math.toRadians(ângulo));
    // Aplicamos o sistema de coordenadas.
    g2d.transform(at);
    // Desenhamos o robô na posição 0,0. Primeiro o corpo:
    g2d.setColor(cor);
    g2d.fillRect(-10,-12,20,24);
```



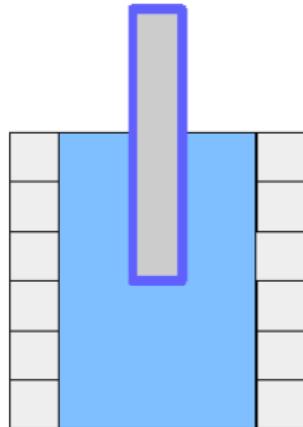
Classe Tanque

```
// Agora as esteiras
for(int e=-12;e<=8;e+=4)
{
    g2d.setColor(Color.LIGHT_GRAY);
    g2d.fillRect(-15,e,5,4);
    g2d.fillRect(10,e,5,4);
    g2d.setColor(Color.BLACK);
    g2d.drawRect(-15,e,5,4);
    g2d.drawRect(10,e,5,4);
}
// Finalmente o canhão.
g2d.setColor(Color.LIGHT_GRAY);
g2d.fillRect(-3,-25,6,25);
g2d.setColor(cor);
g2d.drawRect(-3,-25,6,25);
```



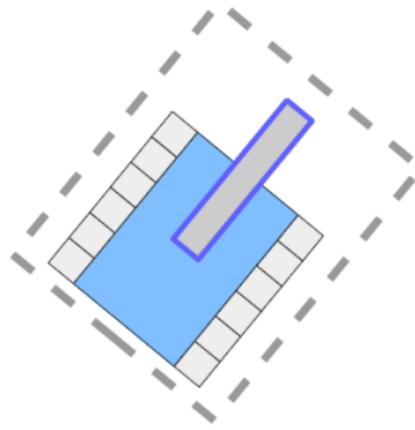
Classe Tanque

```
// Se o tanque estiver ativo, desenhamos uma margem  
nele.  
if (estáAtivo)  
{  
    g2d.setColor(new Color(120,120,120));  
    Stroke linha = g2d.getStroke();  
    g2d.setStroke(new BasicStroke(1f,BasicStroke.  
        CAP_ROUND,  
        BasicStroke.JOIN_ROUND,0,  
        new float[]{8},0));  
    g2d.drawRect(-24,-32,48,55);  
    g2d.setStroke(linha);  
}  
// Aplicamos o sistema de coordenadas original.  
g2d.setTransform(anter);  
}
```



Classe Tanque

```
public Shape getRectEnvolvente()
{
    AffineTransform at = new AffineTransform();
    at.translate(x,y);
    at.rotate(Math.toRadians(ângulo));
    Rectangle rect = new Rectangle(-24,-32,48,55);
    return at.createTransformedShape(rect);
}
```



Classe Arena

```
package tanques;
import java.awt.*;
import java.awt.event.*;
import java.util.HashSet;
import javax.swing.*;
public class Arena extends JComponent implements MouseListener, ActionListener
{
    private int w,h;
    private HashSet<Tanque> tanques;
    private Timer timer;
    public Arena(int w,int h)
    {
        this.w = w; this.h = h;
        tanques = new HashSet<Tanque>();
        addMouseListener(this);
        /*Timer—javax.swing —> dispara um ou mais ActionEvent em intervalo
         especificado em milisegundos — invocando o actionPerformed da classe*/
        timer = new Timer(500,this);
        timer.start();
    }
}
```

Classe Arena

```
public void adicionaTanque(Tanque t)
{
    tanques.add(t);
}
public Dimension getMaximumSize()
{
    return getPreferredSize();
}
public Dimension getMinimumSize()
{
    return getPreferredSize();
}
public Dimension getPreferredSize()
{
    return new Dimension(w,h);
}
```

Classe Arena

```
protected void paintComponent(Graphics g)
{
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.setColor(new Color(245,245,255));
    g2d.fillRect(0,0,w,h);
    g2d.setColor(new Color(220,220,220));
    for(int _w=0;_w<=w;_w+=20) g2d.drawLine(_w,0,_w,h);
    for(int _h=0;_h<=h;_h+=20) g2d.drawLine(0,_h,w,_h);
    // Desenhamos todos os tanques
    for(Tanque t:tanques) t.draw(g2d);
}
```

Classe Arena

```
public void mouseClicked(MouseEvent e)
{
    for(Tanque t:tanques) t.setEstáAtivo(false);
    for(Tanque t:tanques)
    {
        boolean clicado = t.getRectEnvolvente().contains(e.getX(),e.getY());
        if (clicado)
        {
            t.setEstáAtivo(true);
            switch(e.getButton())
            {
                case MouseEvent.BUTTON1: t.giraAntiHorário(3); break;
                case MouseEvent.BUTTON2: t.aumentaVelocidade(); break;
                case MouseEvent.BUTTON3: t.giraHorário(3); break;
            }
            break;
        }
    }
    repaint();
}
```

Classe Arena

```
public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mousePressed(MouseEvent e) { }

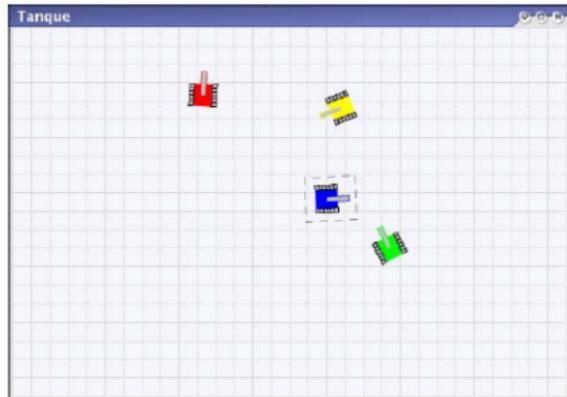
public void mouseReleased(MouseEvent e) { }

public void actionPerformed(ActionEvent e)
{
    for(Tanque t:tanques) t.move();
    repaint();
}
```

Classe App

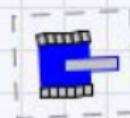
```
package tanques;
import java.awt.Color;
import javax.swing.JFrame;
public class App
{
    public static void main(String[] args)
    {
        Arena arena = new Arena(600,400);
        arena.adicionaTanque(new Tanque(100,200, 0,Color.BLUE));
        arena.adicionaTanque(new Tanque(200,200, 45,Color.RED));
        arena.adicionaTanque(new Tanque(470,360, 90,Color.GREEN));
        arena.adicionaTanque(new Tanque(450, 50,157,Color.YELLOW));
        JFrame f = new JFrame("Tanques");
        f.getContentPane().add(arena);
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Exemplo - simulação simples



Exemplo - simulação simples

Tanque



Acrescentar os seguintes itens à aplicação:

- ① Aumentar a quantidade de tanques na arena;
- ② Adicionar o comportamento “andar de ré”. Utilize o evento de tecla pressionada + botão do mouse. Para isso a aplicação deverá tratar eventos do teclado.
- ③ Adicionar o comportamento “girar em torno do próprio eixo” .
- ④ Acelerar os tanques
- ⑤ Desacelerar os tanques
- ⑥ Não permitir que os tanques ultrapassem os limites da arena.
- ⑦ Adicionar outras propriedades e/ou comportamentos de acordo com sua criatividade.
- ⑧ Gerar o arquivo Jar da aplicação.