

1. Classes

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



Classe Retângulo

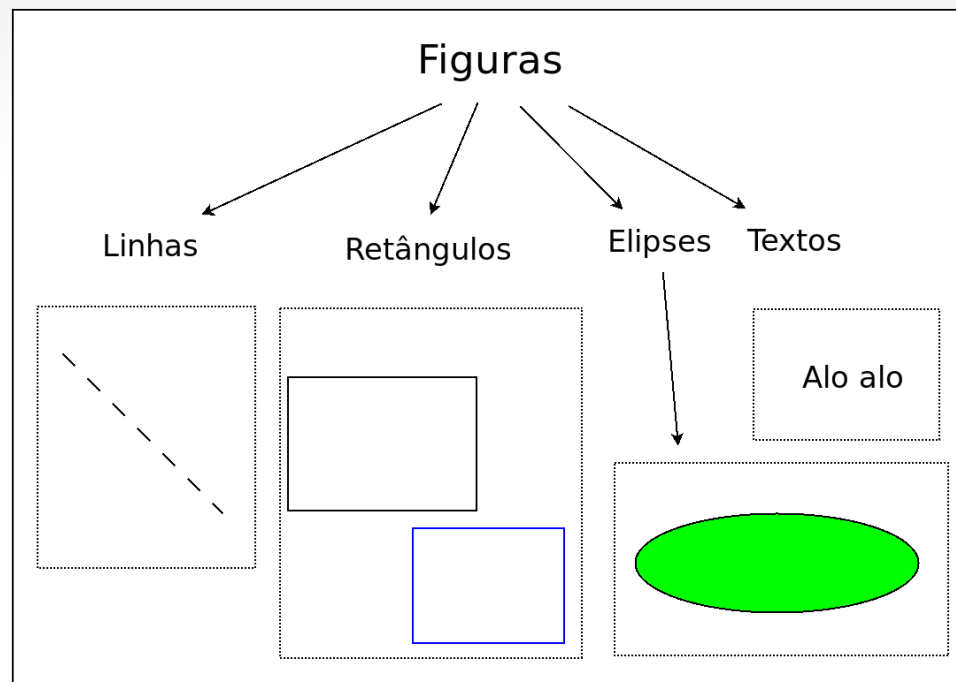
- Propriedades
 - (x, y) : Posição
 - (w, h) : Tamanho

C

```
typedef struct {  
    int x, y;  
    int w, h;  
} Rect;
```

Java

```
class Rect {  
    int x, y;  
    int w, h;  
}
```



Retângulos - C e Java

“Exibir” Retângulo

```
typedef struct {  
    int x, y;  
    int w, h;  
} Rect;
```

C

```
void print (Rect* r) {  
    printf (  
        "Tam (%d,%d) / Pos (%d,%d)\n",  
        r->w, r->h, r->x, r->y  
    );  
}
```

```
void main (void) {  
    Rect r1 = { 1,1, 10,10 };  
    print(&r1);  
}
```

```
class Rect {  
    int x, y;  
    int w, h;  
}
```

Java

```
class Rect {  
    ...  
    void print () {  
        System.out.format (  
            "Tam (%d,%d) / Pos (%d,%d)\n",  
            this.w, this.h, this.x, this.y  
        );  
    }  
}
```

```
public class RectApp {  
    public static void main (...) {  
        Rect r1 = new Rect(1,1,10,10);  
        r1.print();  
    }  
}
```

Classes

- Propriedades
- Construtor
- Métodos
- Instância
- Uso

```
class Rect {  
    int x, y;  
    int w, h;  
  
    Rect (int x, int y, int w, int h) {  
        this.x = x;  
        this.y = y;  
        this.w = w;  
        this.h = h;  
    }  
  
    void print () {  
        System.out.format("(%d,%d) / (%d,%d)\n",  
            this.w, this.h, this.x, this.y);  
    }  
}
```

```
public class RectApp {  
    public static void main (String[] args) {  
        Rect r1 = new Rect(1,1, 10,10);  
        r1.print();  
    }  
}
```

Classes

- Retângulo

```
class Rect {  
    int x, y;  
    float rot;  
    int r, g, b;  
    int w, h;  
    ...  
}
```

- Texto

```
class Text {  
    int x, y;  
    float rot;  
    int r, g, b;  
    int size;  
    String face;  
    ...  
}
```

Exercícios

1. Considere as 2 figuras mais complexas do exercício anterior:

1. Implemente uma struct em C para uma delas e uma classe em Java para a outra.
2. Implemente uma função em C e um método em Java "print" para elas.

2. Considere a classe de Retângulos e implemente os seguintes métodos:

1. `int area ()`
 - deve retornar a área do retângulo
2. `void drag (int dx, int dy)`
 - deve “arrastar” o objeto, ou seja somar o dx e dy a sua posição atual

1. Classes

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



2. Painting

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



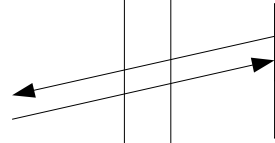
Painting

- Adicionar método `paint` ao retângulo
- Adicionar nova classe `Ellipse` com `paint`
- Criar aplicação com retângulos e elipses

Estado Atual

```
class Rect {  
    int x, y;  
    int w, h;  
    Rect (int x, int y, int w, int h) {  
        this.x = x;  
        this.y = y;  
        this.w = w;  
        this.h = h;  
    }  
    void print () {  
        System.out.format(...);  
    }  
    void paint (Graphics g) { ... }  
}
```

```
class Hello2DFrame extends JFrame {  
    public Hello2DFrame () {  
        this.setTitle("Hello World!");  
        this.setSize(350, 350);  
    }  
    public void paint (Graphics g) {  
        super.paint(g);  
        Graphics2D g2d = (Graphics2D) g;  
        g2d.setPaint(Color.blue);  
        int w = this.getWidth();  
        int h = this.getHeight();  
        r.paint(g);  
    }  
}
```



PaintApp

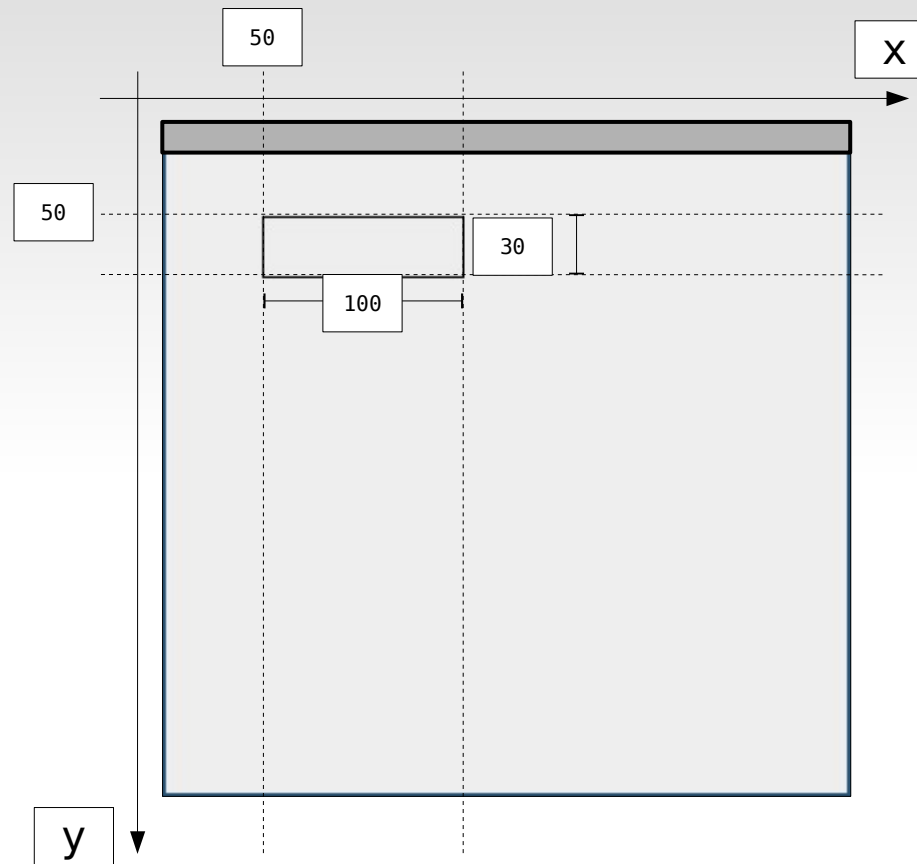
```
class Rect {  
    int x, y;  
    int w, h;  
    ...  
    void paint (Graphics g) {  
        Graphics2D g2d = ...  
        g2d.drawRect (  
            this.x, this.y,  
            this.w, this.h  
        );  
    }  
}
```

Faça algumas modificações no programa:

- adicione propriedades de cor de contorno e cor de fundo
- altere o método paint para desenhar a cor de contorno e cor de fundo do retângulo
- crie pelo menos 3 retângulos com propriedades diferentes e os exiba na tela

```
class PaintApp {  
    public static void main (String[] args) {  
        PaintFrame frame = new PaintFrame();  
        frame.setVisible(true);  
    }  
}  
  
class PaintFrame extends JFrame {  
    Rect r1;  
    PaintFrame () {  
        this.setTitle("Painting Figures");  
        this.setSize(350, 350);  
        this.r1 = new Rect(50,50, 100,30);  
    }  
    public void paint (Graphics g) {  
        super.paint(g);  
        this.r1.paint(g);  
    }  
}
```

Sistema de Coordenadas



```
//           x   y   w   h  
this.r1 = new Rect(50,50, 100,30);
```

Elipses

```
class Ellipse {  
    int x, y;  
    int w, h;  
    ...  
    void paint (Graphics g) {  
        Graphics2D g2d = ...  
        g2d.draw (  
            new Ellipse2D.Double(  
                this.x, this.y,  
                this.w, this.h  
            )  
        );  
    }  
}
```

Pacotes (packages)

```
class RectEllipseApp {  
    public static void main (String[] args) {  
        ...  
    }  
}  
  
class RectEllipseFrame extends JFrame {  
    ...  
    this.r1 = new Rect(50,50, 100,30);  
    this.e1 = new Ellipse(50,100, 100,30);  
}  
    public void paint (Graphics g) {  
        super.paint(g);  
        this.r1.paint(g);  
        this.e1.paint(g);  
    }  
}
```

Faça as mesmas modificações pedidas para a classe Rect.

04-Paint/

2. Painting

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



3. Pacotes e Visibilidade (Java)

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



Packages

- Organizam as classes em diretórios e arquivos
- Diretivas **package** e **import**
- Classes e métodos exportados devem ser **public**
- Cada arquivo com somente uma classe pública

```
import figures.Rect;
import figures.Ellipse;
class PackApp {
    ...
}
class PackFrame extends JFrame {
    ...
    PackFrame () {
        this.r1 = new Rect(...);
        this.e1 = new Ellipse(...);
    }
}
```

```
package figures;
public class Rect {
    public Rect (...) {
        ...
    }
    public void print () {
        ...
    }
    public void paint (Graphics g) {
        ...
    }
}
```

Modificadores de Acesso

- Classes, métodos e propriedades
- *nenhum*: visível somente no pacote
- **private**: visível somente na classe
- **protected**: visível no pacote e subclasses
- **public**: visível por todos

Evitar o modificador **public**

- a) Não usar nenhum modificador até que se faça necessário.
- b) Usar o modificador **private** até que se faça necessário.

Modificadores de Acesso

```
package figures;
import java.awt.*;
public class Rect {
    private int x, y;
    private int w, h;
    public Rect (int x, int y, int w, int h) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }
    public void print () {
        ...
    }
    public void paint (Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawRect(this.x, this.y, this.w, this.h);
    }
}
```

- Adicione uma nova figura ao projeto:

- a) crie uma nova classe no pacote de figuras
- b) use os modificadores de acesso apropriados
- c) adicione novas figuras ao programa principal

Adapte as suas modificações anteriores (ex., cores de contorno e fundo) para estarem nas classes divididas em pacotes e com os modificadores de acesso apropriados.

3. Pacotes e Visibilidade (Java)

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



4. Listas de Classes

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



Listas Heterogêneas

- Em uma mesma lista, guardar retângulos e elipses.
 - `figs = { r1, e1, r2, e2, ... }`
 - `figs[i].paint()`
- Python e C

O que significa “duck typing” e como esse conceito se relaciona com o que queremos fazer com listas heterogêneas?

```
int l[3] = { 1, "ola", 3 };  
int v = l[0] + l[1];
```

```
ArrayList<Figure> figs = new ArrayList<Figure>();  
figs.add(  
figs.add(  
for (Fig  
    fig.  
}
```

Subtipagem (Subclassing)

Superclasse abstrata **Figure** com subclasses concretas **Rect** e **Ellipse**

Nova demanda!

- Adicionar um novo retângulo sempre que o usuário pressionar a tecla “r”
 - criar com posição e tamanho aleatórios
- Vamos precisar manter uma lista de retângulos:
 - `ArrayList<Rect> rs = new ArrayList<Rect>();`
 - `rs.add(new Rect(x,y, w,h));`
 - ```
for (Rect r: this.rs) {
 r.paint(g);
}
```
- Como detectar que a tecla “r” foi pressionada?

# Simplificado

```
class SimpleFrame extends JFrame {
 ArrayList<Rect> rs = new ArrayList<Rect>();
 SimpleFrame () {
 Random rand = new Random();
 for (int i=0; i<4; i++) {
 int x = rand.nextInt(350);
 int y = rand.nextInt(250);
 rs.add(new Rect(x,y, w,h));
 }
 }
 public void paint (Graphics g) {
 super.paint(g);
 for (Rect r: this.rs) {
 r.paint(g);
 }
 }
}
```

Como detectar que a tecla "r" foi pressionada?

06-List/SimpleApp.java



## 4. Listas de Classes

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## 5. *Event Listeners*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Event Listeners

- A classe `JFrame` pode “escutar” eventos do usuário:

```
class ListFrame extends JFrame {
 ListFrame () {
 this.addKeyListener (
 new KeyAdapter() {
 public void keyPressed (KeyEvent evt) {
 if (evt.getKeyChar() == 'r') {
 ...
 }
 }
 }
);
 }
}
```

# ListApp.java

```
class ListFrame extends JFrame {
 ListFrame () {
 this.addKeyListener (
 new KeyAdapter() {
 public void keyPressed (KeyEvent evt) {
 if (evt.getKeyChar() == 'e') {
 int n = rand.nextInt(50);
 rs.add(new Rect(x,y, w,h));
 repaint(); // outer.repaint()
 }
 }
 }
);
 }
}
```

- Faça o mesmo tratamento para Elipses:
  - a) declare uma lista de elipses em separado
  - b) crie uma nova ellipse sempre que “e” for pressionado
  - c) redesenhe todas as elipses na tela

## 5. *Event Listeners*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## *6. Subclassing*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Subclassing

- Superclasse vs Subclasse
- Classe abstrata vs Classe concreta
- Overriding de métodos
- Polimorfismo e *Dynamic Dispatching*
- Herança

# Listas Heterogêneas

- Em uma mesma lista, guardar retângulos e elipses.
  - `figs = { r1, e1, r2, e2, ... }`
  - `figs[i].paint()`

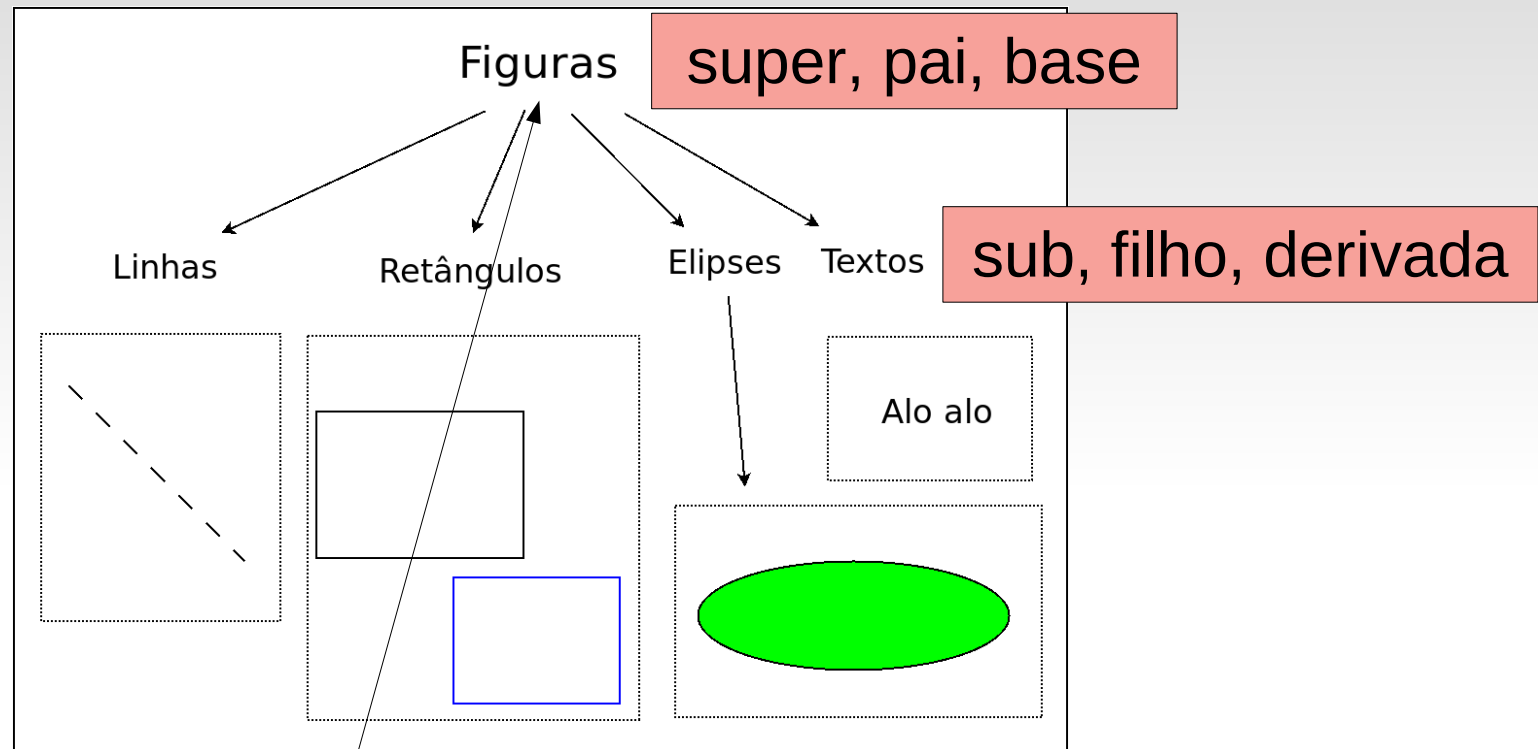
```
ArrayList<Figure> figs = new ArrayList<Figure>();
figs.add(new Rect(x1,y1, w1,h1));
figs.add(new Ellipse(x2,y2, w2,h2));
for (Figure fig: figs) {
 fig.paint(g);
}
```

## Subtipagem (Subclassing)

Superclasse abstrata **Figure** com  
subclasses concretas **Rect** e **Ellipse**



# Figuras



```
ArrayList<Figure> figs = ...
figs.add(new Rect(x1,y1, w1,h1));
figs.add(new Ellipse(x2,y2, w2,h2));
for (Figure fig: figs) {
 fig.paint(g);
}
```

# Superclasse vs Subclasse

- Superclasse `Figure`
  - Abstrata: não implementa o método `paint`
- Subclasse `Rect`
  - Concreta: implementa (overrides) `paint`

```
package figures;
public abstract class Figure {
 public abstract
}
```

- Adapte o código com duas listas para uma só:

- a) crie o tipo abstrato `Figure`
- b) use uma lista única de figuras
- c) torne retângulos, elipses, etc como sendo figuras
- d) crie novas instâncias das figuras reagindo às teclas

```
package figures;
public class Rect extends Figure {

 (Graphics g) {
 (Graphics2D) g;
);
}
```

# Polimorfismo de Subtipos

- Mesmo símbolo, múltiplos sentidos

```
for (Figure fig: figs) {
 fig.paint(g);
}
```

```
public void draw(Graphics2D g2d) {
 g2d.drawRect(...);
}
```

- Procure exemplos de polimorfismo com despacho dinâmico em outros projetos seus ou de terceiros:

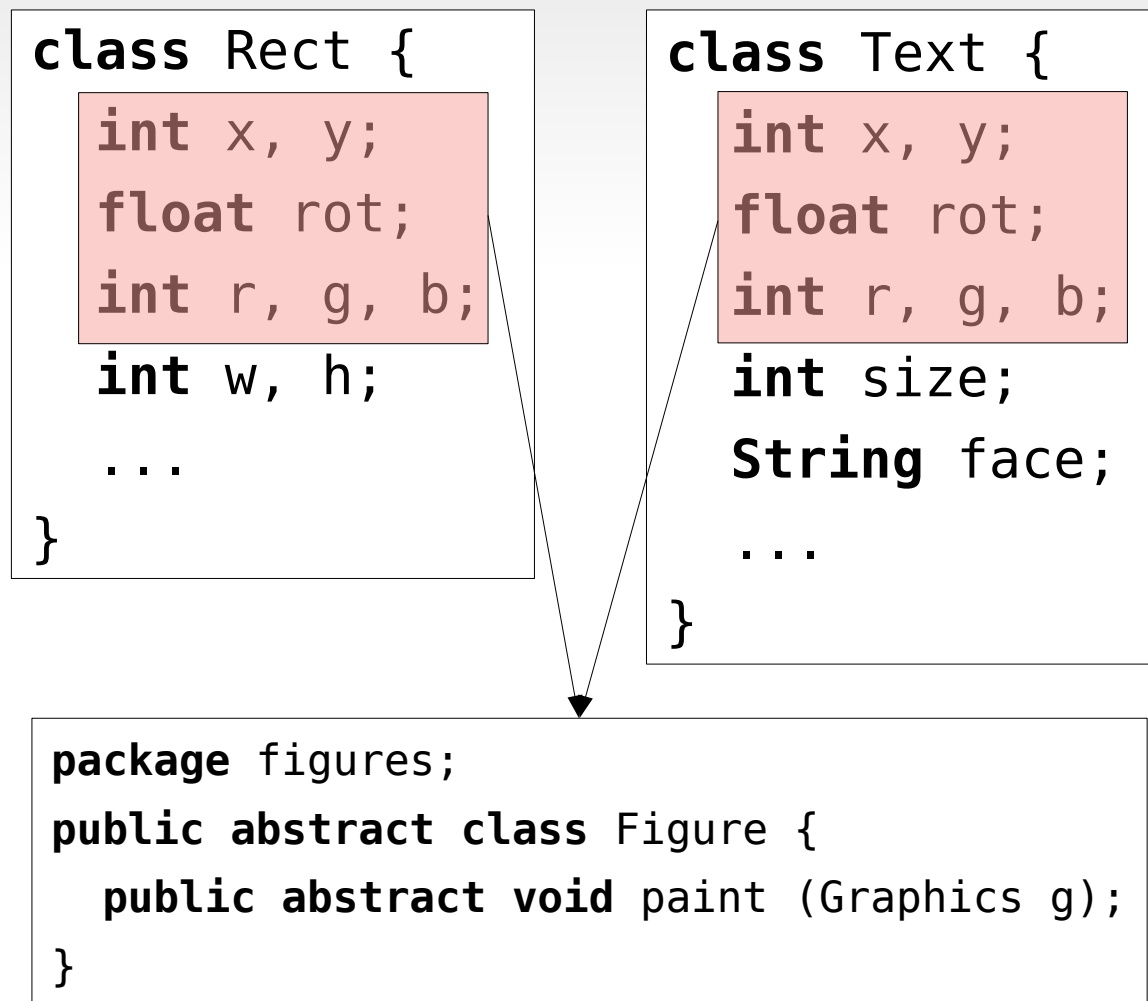
- a) Mostre os trechos de código relevantes.
- b) Explique como se dá o polimorfismo.

```
Graphics g) {
 Graphics2D g;
 g2d(...);
}
```

- Impossível determinar à priori
  - decisão de quem chamar ocorre em execução
  - ***Dinamic Dispatching***

# Herança

- Subclasses podem *herdar* métodos e propriedades

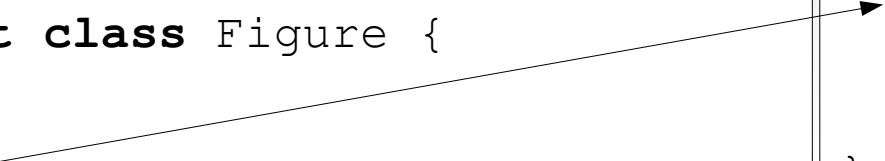


# Herança

- Subclasses herdam com `extends`

```
package figures;
public abstract class Figure {
 int x, y;
 float rot;
 int r, g, b;
 public abstract void paint (Graphics g);
}
```

```
class Rect extends Figure {
 int w, h;
 ...
}
```

A thin grey arrow originates from the `Figure` class name in the `Rect` class definition and points to the `Figure` class definition in the adjacent code block, illustrating the inheritance relationship.

Refatore o seu projeto de modo a concentrar na classe `Figure` os métodos e propriedades em comum entre todas as figuras.

# Subclassing

- Superclasse vs Subclasse
- Classe abstrata vs Classe concreta
- Overriding de métodos
- Polimorfismo e *Dynamic Dispatching*
- Herança

## 6. Subclassing

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## 7. Comparação com C

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br





# Subclassing

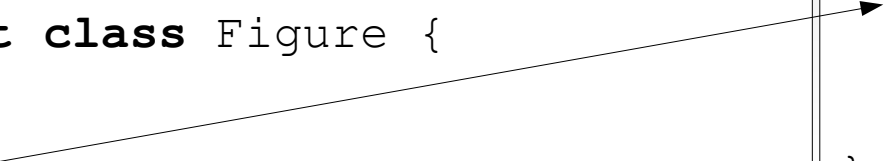
- Superclasse vs Subclasse
- Classe abstrata vs Classe concreta
- Overriding de métodos
- Polimorfismo e *Dynamic Dispatching*
- Herança

# Herança

- Subclasses herdam com `extends`

```
package figures;
public abstract class Figure {
 int x, y;
 float rot;
 int r, g, b;
 public abstract void paint (Graphics g);
}
```

```
class Rect extends Figure {
 int w, h;
 ...
}
```

A thin black arrow originates from the `Figure` class name in the `Rect` class definition and points to the `Figure` class definition in the adjacent code block, illustrating the inheritance relationship.

# Polimorfismo

- Mesmo símbolo, múltiplos sentidos

```
for (Figure fig: figs) {
 fig.paint(g);
}
```

```
public void paint (Graphics g) {
 Graphics2D g2d = (Graphics2D) g;
 g2d.drawRect(...);
}
```

```
public void paint (Graphics g) {
 Graphics2D g2d = (Graphics2D) g;
 g2d.draw(new Ellipse2D(...));
}
```

- Impossível determinar à priori
  - decisão de quem chamar ocorre em execução
  - ***Dynamic Dispatching***

# Classe Retângulo

## ■ Propriedades

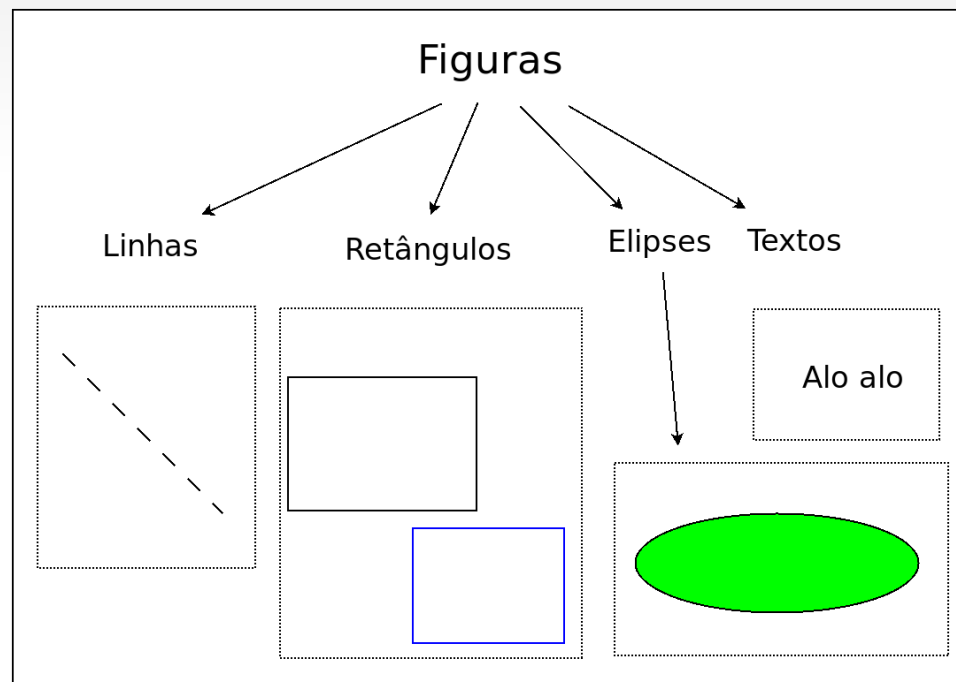
- $(x, y)$ : Posição
- $(w, h)$ : Tamanho

C

```
typedef struct {
 int x, y;
 int w, h;
} Rect;
```

Java

```
class Rect {
 int x, y;
 int w, h;
}
```



# “Exibir” Retângulo

```
typedef struct {
 int x, y;
 int w, h;
} Rect;
```

C

```
void print (Rect* r) {
 printf (
 "Tam (%d,%d) / Pos (%d,%d)\n",
 r->w, r->h, r->x, r->y
);
}
```

```
void main (void) {
 Rect r1 = { 1,1, 10,10 };
 print(&r1);
}
```

```
class Rect {
 int x, y;
 int w, h;
}
```

Java

```
class Rect {
 ...
 void print () {
 System.out.format (
 "Tam (%d,%d) / Pos (%d,%d)\n",
 this.w, this.h, this.x, this.y
);
 }
}
```

```
public class RectApp {
 public static void main (...) {
 Rect r1 = new Rect(1,1,10,10);
 r1.print();
 }
}
```

# Herança e Polimorfismo

- Subclasses herdam com `extends` e implementam métodos abstratos com `overriding`

```
abstract class Figure {
 int x, y;
 int r, g, b;
 abstract void print (void);
}
```

```
class Rect extends Figure {
 int w, h;
 ...
 void print (void) {...}
}
```

```
for (Figure fig: figs) {
 fig.print();
}
```

extends  
overriding

# Herança e Polimorfismo em C

- Structs aninhadas (*extends*)
- Ponteiros para funções (*overrides*)

```
abstract class Figure {
 int x, y;
 int r, g, b;
 abstract void print (void);
}
```

```
class Rect extends Figure {
 int w, h;
 ...
 void print (void) {...}
}
```

```
typedef struct Figure {
 int x, y;
 int r, g, b;
 void (* print) (struct Figure*);
} Figure;
```

```
typedef struct Rect {
 Figure super;
 int w, h;
 ...
} Rect;
```

```

typedef struct Figure {
 int x, y;
 int r, g, b;
 void (* print) (struct Figure*);
} Figure;

```

```

typedef struct Rect {
 Figure super;
 int w, h;
 ...
} Rect;

```

```

void main (void) {
 Figure* figs[N] = {
 (Figure*) rect_new(10,10,...),
 (Figure*) ellipse_new(40,...),
 ...
 };
 /////
 for (int i=0; i<N; i++) {
 figs[i]->print(figs[i]);
 }
 /////
 for (int i=0; i<N; i++) {
 free(figs[i]);
 }
}

```

```

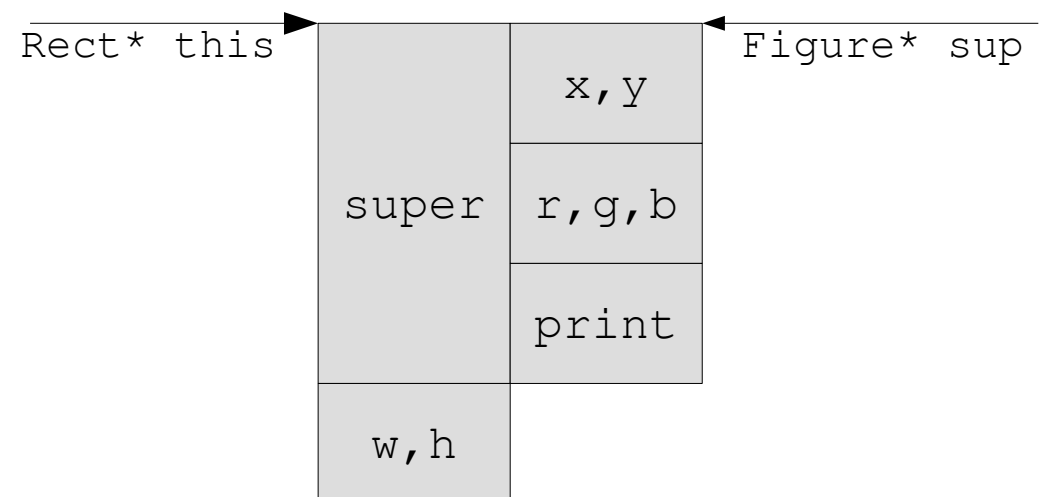
void rect_print (Rect* this) {
 Figure* sup = (Figure*) this;
 printf("Retangulo (%d,%d) ...\n",
 sup->x, sup->y, ...);
}

```

```

Rect* rect_new (int x, int y, ...) {
 Rect* this = malloc(sizeof(Rect));
 Figure* sup = (Figure*) this;
 sup->print = rect_print;
 sup->x = x;
 sup->y = y;
 ...
 return this;
}

```





# Exercício

- Converta algumas classes do projeto para C:
  - Uma *struct* para representar a classe Figure.
  - Duas *structs* para as duas figuras mais complexas do seu projeto.
  - A main deve criar algumas figuras e exibí-las na tela.
- Instruções extras:
  - Troque o método `draw` por `print`.
  - Baseie-se no código de exemplo:
    - <https://github.com/fsantanna-uerj/LP2/tree/main/Exemplos/08-Figure-C>

# *7. Comparação com C*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## 8. *Virtual Method Table (vtable)*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Virtual Method Table (vtable)

- Em vez de ter 1 ponteiro pra cada um dos 12 métodos...

```
abstract class Figure {
 int x, y;
 int r, g, b;
 abstract void print (void);
 abstract int area (void);
 ... // 10 outros métodos
}
```

```
typedef struct Figure {
 int x, y;
 int r, g, b;
 void (* print) (struct Figure*);
 int (* area) (struct Figure*);
 ... // 10 outros ponteiros
} Figure;
```

```
Rect* rect_new (int x, int y, ...) {
 Rect* this = malloc(sizeof(Rect));
 ...
 super->print = rect_print;
 super->area = rect_area;
 ...
 return this;
}
```

12 \* 8 = 96 bytes  
por objeto

# Virtual Method Table (vtable)

- ... usar ponteiro único para tabela com 12 ponteiros.

```
typedef struct Figure {
 int x, y;
 int r, g, b;
 void (* print) (struct Figure*);
 int (* area) (struct Figure*);
 ... // 10 outros ponteiros
} Figure;
```

```
typedef struct {
 void (* print) (struct Figure*);
 int (* area) (struct Figure*);
 ... // 10 outros ponteiros
} Figure_vtable;
```

```
typedef struct Figure {
 int x, y;
 int r, g, b;
 Figure_vtable* vtable;
} Figure;
```

8 bytes  
por objeto

# Virtual Method Table (vtable)

- ... usar ponteiro único para tabela com 12 ponteiros.

```
typedef struct Figure {
 int x, y;
 int r, g, b;
 Figure_vtable* vtable;
} Figure;
```

```
Figure_vtable rect_vtable = {
 rect_print,
 rect_area,
 ...
};
```

- 8 bytes por objeto
- acesso indireto

```
Rect* rect_new (int x, int y, ...) {
 Rect* this = malloc(sizeof(Rect));
 ...
 super->vtable = &rect_vtable;
 ...
 return this;
}
```

```
void main (void) {
 ...
 for (int i=0; i<N; i++) {
 figs[i]->vtable->print(figs[i]);
 }
 ...
}
```

# Exercício

- Converta algumas classes do projeto para C:
  - Uma *struct* para representar a classe Figure.
  - **Pelo menos 3 métodos virtuais (print, area, ?)**
  - Duas *structs* para as duas figuras mais complexas do seu projeto.
  - A main deve criar algumas figuras e usar os métodos virtuais.
- Instruções extras:
  - **Use o mecanismo de vtables para polimorfismo.**
  - Baseie-se no código de exemplo:
    - <https://github.com/fsantanna-uerj/LP2/tree/main/Exemplos/08-Figure-C>
    - `vtable.c`

# Herança e Polimorfismo

- Herança
  - Structs aninhadas (*extends*)
- Polimorfismo
  - Ponteiros para funções (*overrides*)
    - Virtual table (+compacto, -indireção)
- Em C: *unsafe, typecasts*



## 8. *Virtual Method Table (vtable)*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br

