

Programação Orientada por Objetos

Herança de classes - exemplo

Prof. Cédric Grueau

Prof. José Sena Pereira

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal
Instituto Politécnico de Setúbal

2022/2023



Sumário

- ▶ Herança – Exemplo Xadrez
- ▶ Redefinição de métodos
- ▶ Herança – Exemplo Formas Geométricas
- ▶ Generalização versus especialização de classes
- ▶ A Classe **Object**



Exemplo – Xadrez

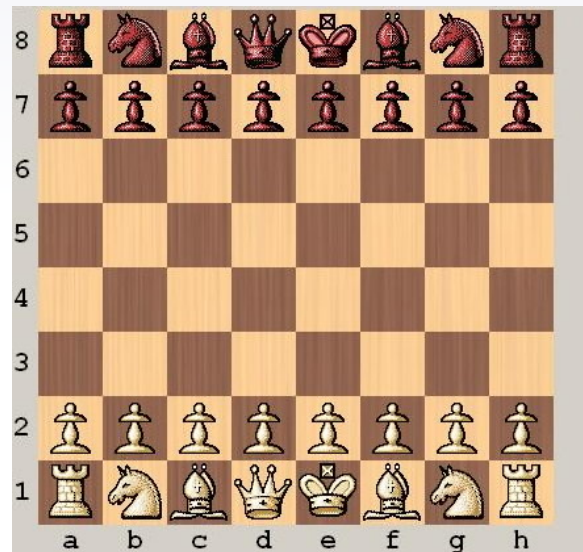
- ▶ Requisitos do protótipo:
 - ▶ Representar os componentes do jogo sem implementar as regras ou o desenrolar do jogo.
 - ▶ Representar as peças: peão, torre, cavalo, rei, rainha e bispo.
 - ▶ Representar o tabuleiro de jogo com as posições.
 - ▶ Deve ser possível obter em texto a posição de cada peça usando a notação algébrica (ex: e5 – peão na casa e5, ou Te7 – torre na casa e7).



Exemplo – Xadrez

Representação 1

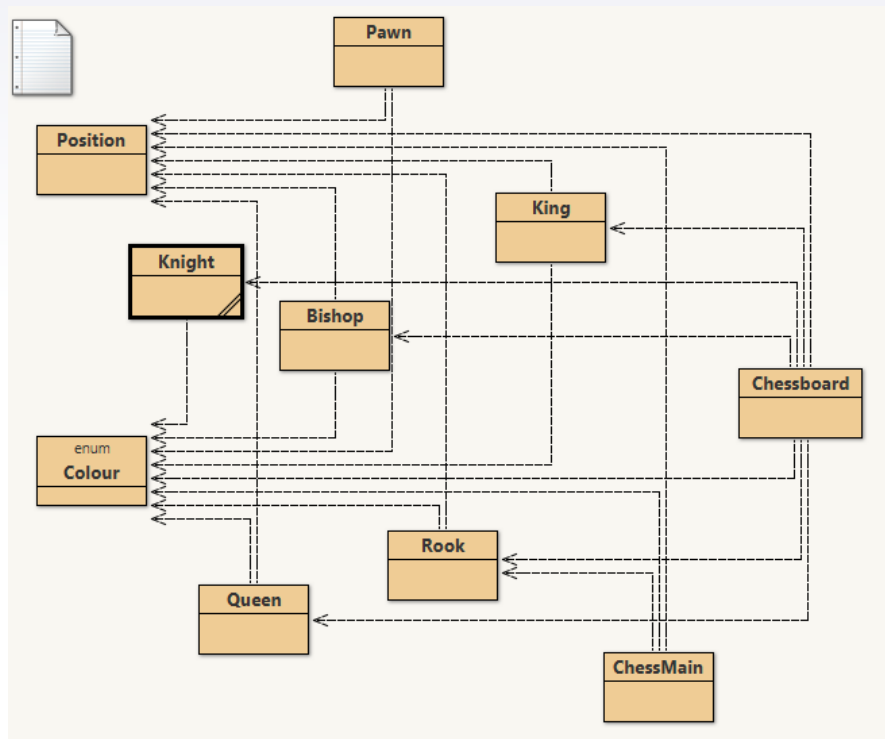
- ▶ Cada peça poderá ser representada por uma classe
- ▶ Peças: peão, torre, cavalo, rei, rainha e bispo.
- ▶ O tabuleiro de jogo corresponde a outra classe.



Exemplo – Xadrez

Representação 1

- ▶ Cada peça poderá ser representada por uma classe
- ▶ Peças: peão, torre, cavalo, rei, rainha e bispo.
- ▶ O tabuleiro de jogo corresponde a outra classe.



Exemplo – Xadrez

► Representação 1 – Classes **Pawn** e **Rook**

```
public class Pawn {

    private Colour colour;
    private Position position;

    public Pawn(Colour colour, Position position) {
        this.colour = colour;
        if (position != null) {
            this.position = position;
        } else {
            this.position = new Position();
        }
    }

    public Colour getColour() { ... }
    public Position getPosition() { ... }
    public void setPosition(char x, int y) { ... }
    public void setPosition(Position position)
    {...}
    public void setY(int y) { ... }
    public char getX() { ... }
    public int getY() { ... }
    public String toString() { ... }
    public String getName() { ... }

}
```

```
public class Rook {

    private Colour colour;
    private Position position;

    public Rook(Colour colour, Position position) {
        this.colour = colour;
        if (position != null) {
            this.position = position;
        } else {
            this.position = new Position();
        }
    }

    public Colour getColour() { ... }
    public Position getPosition() { ... }
    public void setPosition(char x, int y) { ... }
    public void setPosition(Position position)
    {...}
    public void setY(int y) { ... }
    public char getX() { ... }
    public int getY() { ... }
    public String toString() { ... }
    public String getName() { ... }

}
```

Exemplo – Xadrez

► Representação 1 – Classe **ChessBoard**

```
public class Chessboard {  
  
    ArrayList<Pawn> pawns;  
    ArrayList<Knight> knights;  
    ArrayList<Rook> rooks;  
    ArrayList<Queen> queens;  
    ArrayList<Bishop> bishops;  
    ArrayList<King> kings;  
  
    public Chessboard() {  
        pawns = new ArrayList<>();  
        knights = new ArrayList<>();  
        rooks = new ArrayList<>();  
        queens = new ArrayList<>();  
        kings = new ArrayList<>();  
        bishops = new ArrayList<>();  
  
        setup();  
    }  
}
```

```
private void setup() {  
    for (char x = 'a'; x <= 'h'; x++) {  
        pawns.add(new Pawn(Colour.WHITE, new Position(x, 2)));  
        pawns.add(new Pawn(Colour.BLACK, new Position(x, 7)));  
    }  
    int line = 1;  
    Colour colour = Colour.WHITE;  
    rooks.add(new Rook(colour, new Position('a', line)));  
    knights.add(new Knight(colour, new Position('b', line)));  
    bishops.add(new Bishop(colour, new Position('c', line)));  
    queens.add(new Queen(colour, new Position('d', line)));  
    kings.add(new King(colour, new Position('e', line)));  
    bishops.add(new Bishop(colour, new Position('f', line)));  
    knights.add(new Knight(colour, new Position('g', line)));  
    rooks.add(new Rook(colour, new Position('h', line)));  
    line = 8;  
    colour = Colour.BLACK;  
    rooks.add(new Rook(colour, new Position('a', line)));  
    knights.add(new Knight(colour, new Position('b', line)));  
    bishops.add(new Bishop(colour, new Position('c', line)));  
    queens.add(new Queen(colour, new Position('d', line)));  
    kings.add(new King(colour, new Position('e', line)));  
    bishops.add(new Bishop(colour, new Position('f', line)));  
    knights.add(new Knight(colour, new Position('g', line)));  
    rooks.add(new Rook(colour, new Position('h', line)));  
}  
}
```

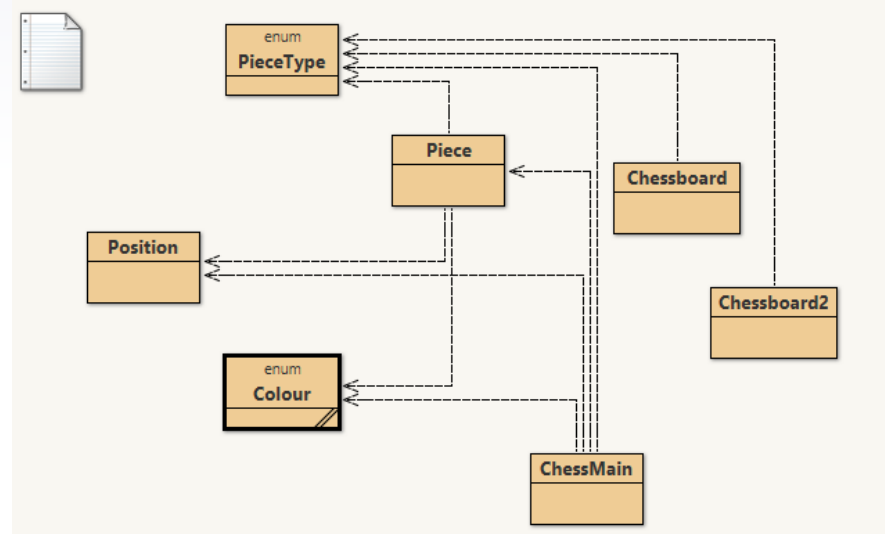
Exemplo – Xadrez

- ▶ Representação 1 – Problemas da solução encontrada:
 - ▶ Duplicação de código nas classes das peças.
 - ▶ A representação do tabuleiro ficou complexa.
 - ▶ Uma lista de peças por tipo de peça (6 listas no total)

Exemplo – Xadrez

Representação 2

- ▶ Criar uma classe peça que representa qualquer peça
- ▶ Usar um atributo **PieceType** que determina qual a peça representada.
- ▶ O tabuleiro de jogo corresponde a outra classe.



Exemplo – Xadrez

► Representação 2 – Classe **Piece**

```
public class Piece {  
  
    private Colour colour;  
    private Position position;  
    private PieceType pieceType;  
  
    public Piece(PieceType pieceType,  
                Colour colour, Position position) {  
        this.pieceType = pieceType;  
        this.colour = colour;  
        if (position != null) {  
            this.position = position;  
        } else {  
            this.position = new Position();  
        }  
    }  
  
    public Colour getColour() { ... }  
    public PieceType getPieceType() { ... }  
    public void setPieceType(PieceType pieceType) { ... }  
    public Position getPosition() { ... }  
    public void setPosition(char x, int y) { ... }  
    public void setPosition(Position position) { ... }  
    public void setY(int y) { ... }  
    public char getX() { ... }  
    public int getY() { ... }  
}
```

```
    public String toString() {  
        String text = "";  
        switch(pieceType){  
            case ROOK:  
                text += 'T';  
                break;  
            case KNIGHT:  
                text += 'C';  
                break;  
            case BISHOP:  
                text += 'B';  
                break;  
            case QUEEN:  
                text += 'D';  
                break;  
            case KING:  
                text += 'R';  
                break;  
        }  
        text += position.toString();  
        return text;  
    }  
  
    public String getName() { ... }  
}
```

Exemplo – Xadrez

► Representação 2 – Classe **ChessBoard**

```
public class Chessboard {  
  
    ArrayList<Piece> pieces;  
  
    public Chessboard() {  
        pieces = new ArrayList<>();  
        setup();  
    }  
}
```

```
private void setup() {  
    for (char x = 'a'; x <= 'h'; x++) {  
        pieces.add(new Piece(PieceType.PAWN, Colour.WHITE, new Position(x, 2)));  
        pieces.add(new Piece(PieceType.PAWN, Colour.BLACK, new Position(x, 7)));  
    }  
    int line = 1;  
    Colour colour = Colour.WHITE;  
    pieces.add(new Piece(PieceType.ROOK, colour, new Position('a', line)));  
    pieces.add(new Piece(PieceType.KNIGHT, colour, new Position('b', line)));  
    pieces.add(new Piece(PieceType.BISHOP, colour, new Position('c', line)));  
    pieces.add(new Piece(PieceType.QUEEN, colour, new Position('d', line)));  
    pieces.add(new Piece(PieceType.KING, colour, new Position('e', line)));  
    pieces.add(new Piece(PieceType.BISHOP, colour, new Position('f', line)));  
    pieces.add(new Piece(PieceType.KNIGHT, colour, new Position('g', line)));  
    pieces.add(new Piece(PieceType.ROOK, colour, new Position('h', line)));  
    line = 8;  
    colour = Colour.BLACK;  
    pieces.add(new Piece(PieceType.ROOK, colour, new Position('a', line)));  
    pieces.add(new Piece(PieceType.KNIGHT, colour, new Position('b', line)));  
    pieces.add(new Piece(PieceType.BISHOP, colour, new Position('c', line)));  
    pieces.add(new Piece(PieceType.QUEEN, colour, new Position('d', line)));  
    pieces.add(new Piece(PieceType.KING, colour, new Position('e', line)));  
    pieces.add(new Piece(PieceType.BISHOP, colour, new Position('f', line)));  
    pieces.add(new Piece(PieceType.KNIGHT, colour, new Position('g', line)));  
    pieces.add(new Piece(PieceType.ROOK, colour, new Position('h', line)));  
}  
}
```

Exemplo – Xadrez

- ▶ Representação 2 – Problemas da solução encontrada:
 - ▶ Classe **Piece** complexa. Tem problemas de coesão.
 - ▶ Na classe **Chessboard** ter-se-á de utilizar vários **switch** sempre que se quiser escolher entre os vários tipos de peça.
 - ▶ Exemplo: na movimentação das peças.

Exemplo – Xadrez

- ▶ Solução:



Usar a herança de classes!

Exemplo – Xadrez

- ▶ Requisitos do protótipo:
 - ▶ Representar os componentes do jogo sem implementar as regras ou o desenrolar do jogo.
 - ▶ Representar as peças: peão, torre, cavalo, rei, rainha e bispo.
 - ▶ Representar o tabuleiro de jogo com as posições.
 - ▶ Deve ser possível obter em texto a posição de cada peça usando a notação algébrica (ex: e5 – peão na casa e5, ou Te7 – torre na casa e7).



Exemplo – Xadrez (3)

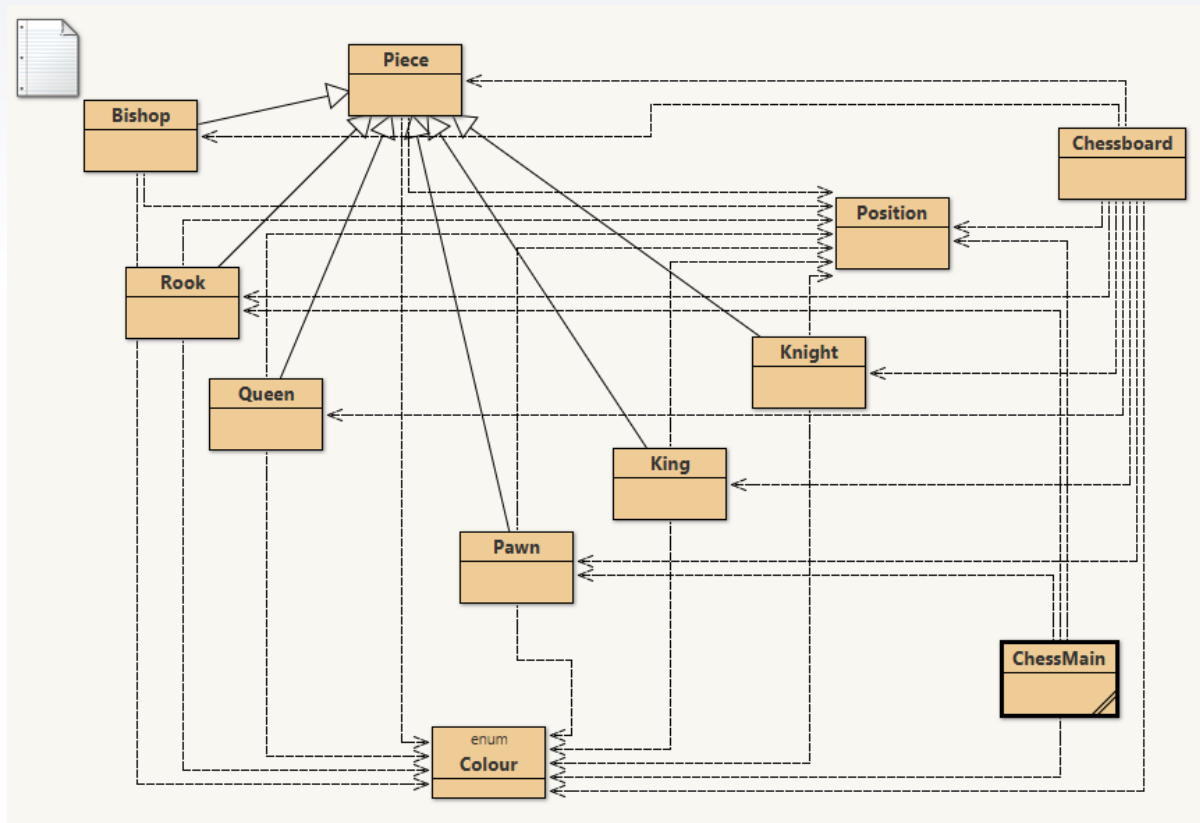
- ▶ Representação 3 – Usar a **herança**:

- ▶ Definir uma classe **Piece** como **superclasse**.
 - ▶ Inclui os atributos e métodos que são idênticos em todas as peças.
- ▶ Definir cada uma das **peças** como uma **subclasse** da classe **Piece**
- ▶ Na classe **Chessboard** ter uma única lista de peças tirando partido do **princípio da substituição**.
 - ▶ Exemplo: na movimentação das peças.



Exemplo – Xadrez (3)

- Solução com herança de classes:




Exemplo – Xadrez (3)

- Exemplo de objetos da representação 1

pawn1 : Pawn

private Colour colour Inspeccionar

private Position position  Obter

Mostrar campos estáticos

Fechar

position : Position

private char x Inspeccionar

private int y Obter

Mostrar campos estáticos

Fechar

pawn1: Pawn


herdado de Object

- Colour getColour()
- String getName()
- Position getPosition()
- char getX()
- int getY()
- void setPosition(char x, int y)
- void setPosition(Position position)
- void setY(int y)
- String toString()



rook1 : Rook

private Colour colour Inspeccionar

private Position position  Obter

Mostrar campos estáticos

Fechar

position : Position

private char x Inspeccionar

private int y Obter

Mostrar campos estáticos

Fechar

rook1: Rook

herdado de Object

- Colour getColour()
- String getName()
- Position getPosition()
- char getX()
- int getY()
- void setPosition(char x, int y)
- void setPosition(Position position)
- void setY(int y)
- String toString()

Exemplo – Xadrez (3)

- ▶ Classe **Piece**

```
public class Piece {  
  
    private Colour colour;  
    private Position position;  
  
    public Piece(Colour colour, Position position)  
    {  
        this.colour = colour;  
        if (position != null) {  
            this.position = position;  
        } else {  
            this.position = new Position();  
        }  
    }  
  
    // restante código omitido  
}
```

Exemplo – Xadrez (3)

- ▶ Classe **Piece** – métodos (1/2)

```
public Colour getColour() {  
    return colour;  
}  
  
public Position getPosition() {  
    return new Position(position.getX(), position.getY());  
}  
  
public void setPosition(char x, int y) {  
    position.setX(x);  
    position.setY(y);  
}  
  
public void setPosition(Position position) {  
    position.setX(position.getX());  
    position.setY(position.getY());  
}
```

Exemplo – Xadrez (3)

- ▶ Classe **Piece** – métodos (2/2)

Podemos ter o método **toString** a retornar o texto da posição

```
public void setY(int y) {  
    position.setY(y);  
}  
  
public char getX() {  
    return position.getX();  
}  
  
public int getY() {  
    return position.getY();  
}  
  
@Override  
public String toString() {  
    return position.toString();  
}
```

Exemplo – Xadrez (3)

- ▶ Classe **Rook**

```
public class Rook extends Piece {
```

Deriva da classe **Piece**

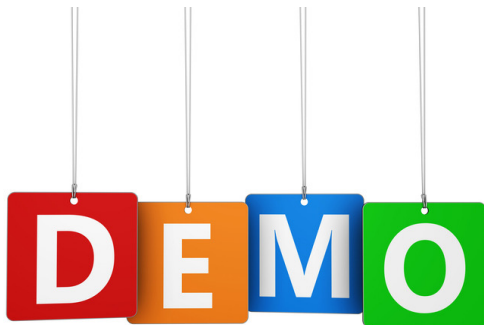
```
    public Rook(Colour colour, Position position) {  
        super(colour, position);  
    }
```

Chamada ao **construtor** da classe **Piece** (superclasse)

```
}
```

Classes **Pawn** e **Rook**

Exemplo – Xadrez (3)



Redefinição de Métodos

- ▶ Herança de classes



Exemplo – Xadrez (3)

► Classe **Rook**

```
public class Rook extends Piece {  
  
    public Rook(Colour colour, Position position) {  
        super(colour, position);  
    }  
  
    public String getName() {  
        return "Torre";  
    }  
  
    @Override  
    public String toString() {  
        return "T" + super.toString();  
    }  
}
```

Deriva da classe **Piece**

Chamada ao construtor da classe **Piece**
(superclasse)

Este método é diferente em todas as classes das
peças

O método **toString** que é herdado
escreve apenas a posição da peça. É
necessário reescrever este método

Exemplo – Xadrez (3)

```
public class Pawn extends Piece{
```

Deriva da classe **Piece**

► Classe **Pawn**

```
    public Pawn(Colour colour, Position position) {  
        super(colour, position);  
    }
```

Chamada ao **construtor** da classe **Piece** (superclasse)

```
    public String getName() {  
        return "Torre";  
    }
```

Este método é diferente em todas as classes das peças

```
    @Override  
    public String toString() {  
        return "P" + super.toString();  
    }
```

O método **toString** que é herdado escreve apenas a posição da peça. É necessário reescrever este método

```
}
```

Herança – Redefinição de métodos

- ▶ Por vezes os **métodos herdados** da superclasse **não servem** nas subclasses porque estão associados a comportamentos próprios das subclasses.
 - ▶ Ex: O método **toString** herdado da classe **Piece** devolve apenas a posição da peça na notação algébrica. Na classe da **Rook** este método deve colocar a letra 'T' antes da posição
- ▶ Neste casos é necessário **redefinir (override)** esse **método**.
 - ▶ A palavra **@Override** que aparece em cima do **toString** quer dizer que o método seguinte é a redefinição de um método que já existe.
- ▶ No entanto é possível **reutilizar os métodos da superclasse** usando o prefixo **super** seguido de um ponto e do identificador do método que se quer utilizar.

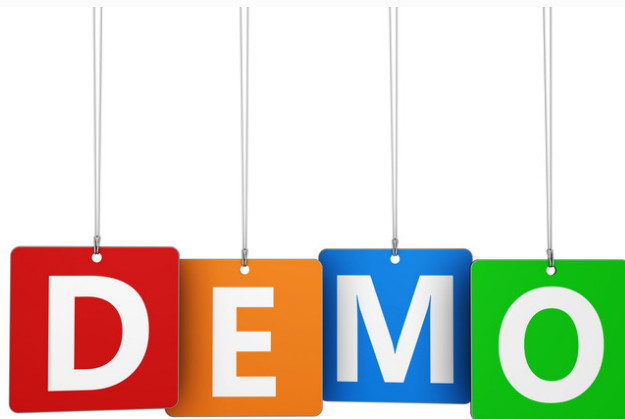
```
@Override  
public String toString() {  
    return "T" + super.toString();  
}
```

Indica que se está a redefinir um método

Chamada ao método **toString** da superclasse

Exemplo – Xadrez (3)

- ▶ Classe **Rook** – Redefinição do método **toString**



Exemplo – Xadrez (3)

- ▶ Classes **Queen, King, Bishop, Knight**
 - ▶ Omitidas: são semelhantes às anteriores
- ▶ Classe **Chessboard**

Apenas uma lista para
guardar as várias peças

```
public class Chessboard {  
  
    ArrayList<Piece> pieces;  
  
    public Chessboard() {  
        pieces = new ArrayList<>();  
        setup();  
    }  
  
    // métodos omitidos  
}
```

Exemplo –

Xadrez (3)

- ▶ Classe **Chessboard** –
método **setup**

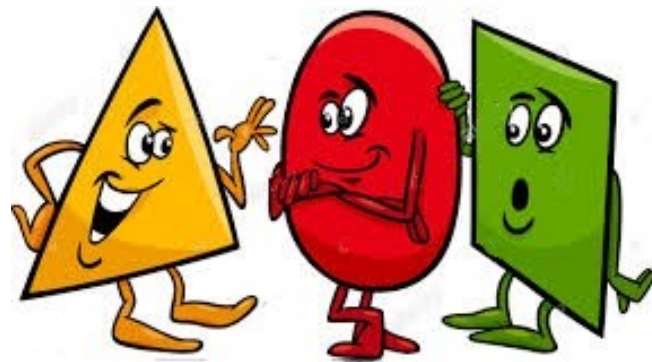
A inicialização tem
de ser feita com o
mesmo detalhe

```
private void setup() {  
    for (char x = 'a'; x <= 'h'; x++) {  
        pieces.add(new Pawn(Colour.WHITE, new Position(x, 2)));  
        pieces.add(new Pawn(Colour.BLACK, new Position(x, 7)));  
    }  
    int line = 1;  
    Colour colour = Colour.WHITE;  
    pieces.add(new Rook(colour, new Position('a', line)));  
    pieces.add(new Knight(colour, new Position('b', line)));  
    pieces.add(new Bishop(colour, new Position('c', line)));  
    pieces.add(new Queen(colour, new Position('d', line)));  
    pieces.add(new King(colour, new Position('e', line)));  
    pieces.add(new Bishop(colour, new Position('f', line)));  
    pieces.add(new Knight(colour, new Position('g', line)));  
    pieces.add(new Rook(colour, new Position('h', line)));  
  
    line = 8;  
    colour = Colour.BLACK;  
    pieces.add(new Rook(colour, new Position('a', line)));  
    pieces.add(new Knight(colour, new Position('b', line)));  
    pieces.add(new Bishop(colour, new Position('c', line)));  
    pieces.add(new Queen(colour, new Position('d', line)));  
    pieces.add(new King(colour, new Position('e', line)));  
    pieces.add(new Bishop(colour, new Position('f', line)));  
    pieces.add(new Knight(colour, new Position('g', line)));  
    pieces.add(new Rook(colour, new Position('h', line)));  
}
```

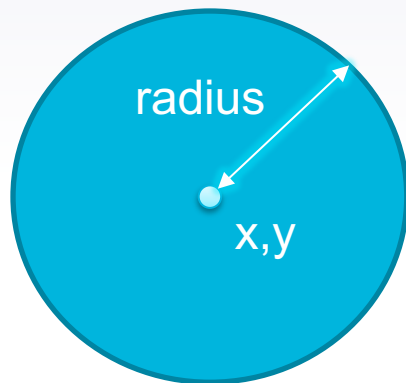
Exemplo – Formas Geométricas

Requisitos do programa:

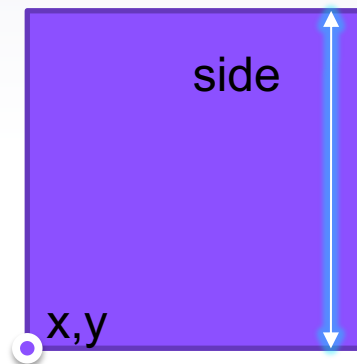
- ▶ Desenho de formas geométricas.
- ▶ Representar apenas círculos e quadrados.
 - ▶ Deve ser possível saber as dimensões e a posição de cada uma delas.
 - ▶ Deve ser possível desloca-los.



Exemplo – Formas Geométricas



Circle



Square

Exemplo – Formas Geométricas

Circle

- x: int
- y : int
- radius : int

+ move()
+ Circle()
+ Circle(int x, int y, int radius)
+ int getX()
+ setX(int x)
+ int getY()
+ setY(int y)
+ int getRadius()
+ void setRadius(int radius)
+ void move(int dx, int dy)

Square

- x: int
- y : int
- side : int

+ move()
+ Square()
+ Square(int x, int y, int side)
+ int getX()
+ void setX(int x)
+ int getY()
+ void setY(int y)
+ int getSide()
+ void setSide(int radius)
+ void move(int dx, int dy)

Exemplo – Formas Geométricas

```
public class Circle {  
    private int x, y;  
    private int radius;  
    public Circle() {  
        this.x = 0;  
        this.y = 0;  
        this.radius = 1;  
    }  
    public Circle(int x, int y, int radius) {  
        this.x = x;  
        this.y = y;  
        this.radius = radius;  
    }  
    public int getRadius() {  
        return radius;  
    }  
    public void setRadius(int raio) {  
        this.radius = radius;  
    }  
}
```

```
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
    public void move( int dx, int dy ) {  
        x += dx; y += dy;  
    }  
}
```

Exemplo – Formas Geométricas

```
public class Square {  
    private int x, y;  
    private int side;  
    public Square() {  
        this.x = 0;  
        this.y = 0;  
        this.side = 1;  
    }  
    public Square(int x, int y, int side) {  
        this.x = x;  
        this.y = y;  
        this.side = side;  
    }  
    public int getSide() {  
        return side;  
    }  
    public void setSide(int side) {  
        this.side = side;  
    }  
}
```

```
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
    public void move( int dx, int dy ) {  
        x += dx; y += dy;  
    }  
}
```

Exemplo –

Formas

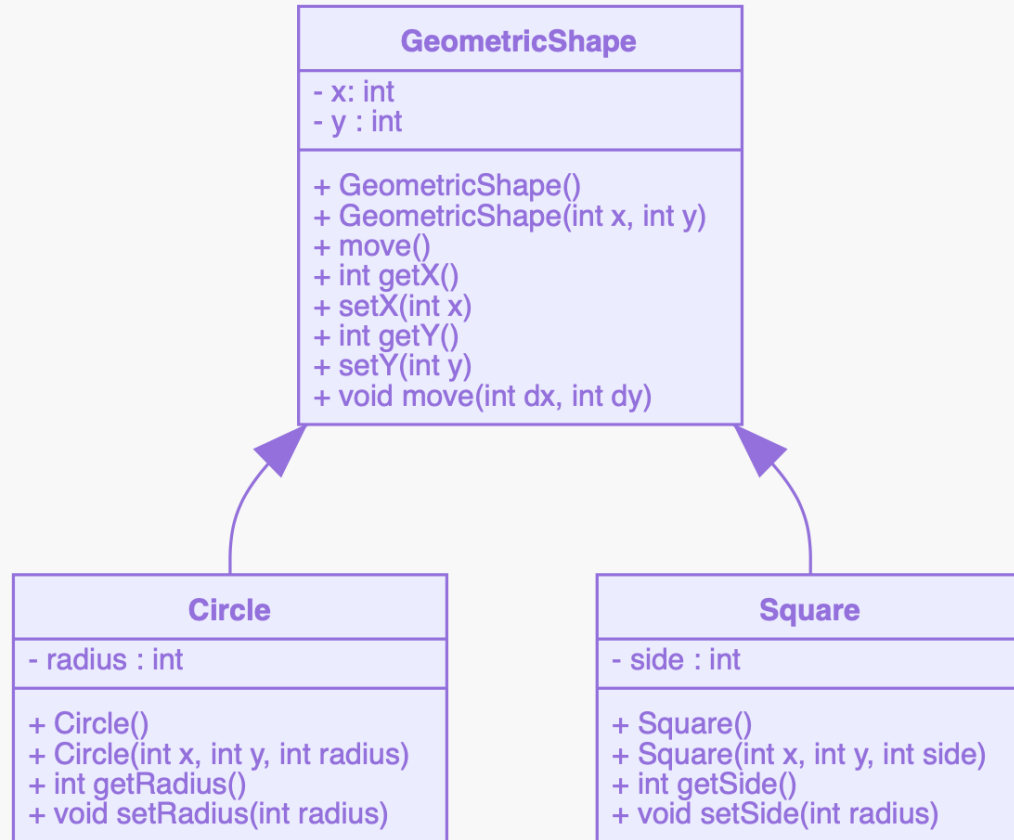
Geométricas

```
public class Program {  
    public static void main(String[] args) {  
        Circle circle = new Circle(1, 1, 23);  
        Square square = new Square(0, 0, 4);  
  
        System.out.println("Circulo: - Posição (" + circle.getX() +  
                            "," + circle.getY() +  
                            ") - Raio: " + circle.getRadius() );  
  
        System.out.println("Quadrado: - Posição (" + square.getX() +  
                            "," + square.getY() +  
                            ") - Lado: " + square.getSide() );  
        square.move( 2, 2);  
  
        System.out.println("Quadrado: - Posição (" + square.getX() +  
                            "," + square.getY() +  
                            ") - Lado: " + square.getLado() );  
    }  
}
```

Exemplo – Formas Geométricas

- ▶ As classes **Circle** e **Square** têm em comum alguns dos atributos e métodos (código duplicado):
 - ▶ 2 atributos (x e y)
 - ▶ 5 getters & setters
- ▶ A solução é utilizar a herança criando uma superclasse **GeometricShape** e definindo **Circle** e **Square** como Subclasses.

Exemplo – Formas Geométricas



Exemplo – Formas Geométricas

```
public class GeometricShape {  
    private int x, y;  
  
    public GeometricShape () {  
        x = 0;  
        y = 0;  
    }  
  
    public GeometricShape (int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

```
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
    public void move( int dx, int dy ) {  
        x += dx; y += dy;  
    }  
}
```

Exemplo – Formas Geométricas

Acrescenta apenas o atributo **radius** e os métodos seletores e modificadores associados

```
public class Circle extends GeometricShape {  
    private int radius;  
    public Circle() {  
        super(0, 0);  
        this.radius = 1;  
    }  
    public Circle(int x, int y, int radius) {  
        super(x, y);  
        this.radius = radius;  
    }  
    public int getRadius() {  
        return radius;  
    }  
    public void setRadius(int radius) {  
        this.radius = radius;  
    }  
}
```

Exemplo – Formas Geométricas

```
public class Square extends GeometricShape {  
    private int side;  
    public Square() {  
        super(0, 0);  
        this.side = 1;  
    }  
    public Square(int x, int y, int side) {  
        super(x, y);  
        this.side = side;  
    }  
    public int getSide() {  
        return side;  
    }  
    public void setSize(int side) {  
        this.side = side;  
    }  
}
```

Acrescenta apenas o atributo **side**
e os métodos seletores e
modificadores associados

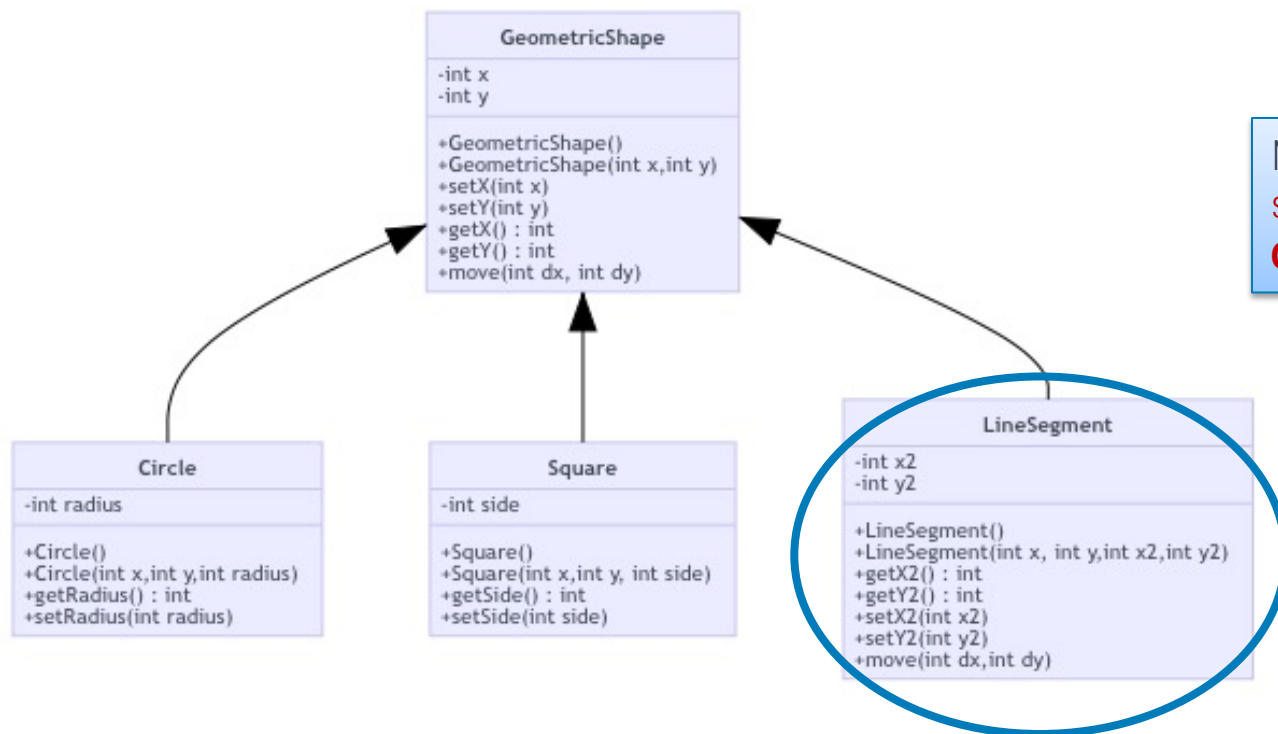
Exemplo – Formas Geométricas

O método main do programa não sofre qualquer alteração

E se quisermos criar
uma nova classe para
representar um
segmento de reta?

```
public class Program {  
    public static void main(String[] args) {  
        Circle circle = new Circle(1, 1, 23);  
        Square square = new Square(0, 0, 4);  
        System.out.println("Circulo: - Posição (" + circle.getX() +  
            "," + circle.getY() +  
            ") - Raio: " + circle.getRadius() );  
        System.out.println("Quadrado: - Posição (" + square.getX() +  
            "," + square.getY() +  
            ") - Lado: " + square.getSide() );  
        square.move( 2, 2);  
        System.out.println("Quadrado: - Posição (" + square.getX() +  
            "," + square.getY() +  
            ") - Lado: " + square.getSide() );  
    }  
}
```

Exemplo – Formas Geométricas



Nova classe definida como
subclasse de
GeometricShape

Exemplo – Formas Geométricas

```
public class LineSegment extends GeometricShape {  
    private int x2, y2;  
    public LineSegment() {  
        super(0, 0);  
        this.x2 = 1;  
        this.y2 = 1;  
    }  
    public LineSegment(int x, int y, int x2, int y2) {  
        super(x, y);  
        this.x2 = x2;  
        this.y2 = y2;  
    }  
}
```

Acrescenta os atributos x2 e y2 para a representação do segundo ponto do segmento de retas

Existe um método que é herdado mas não é adequado nesta classe. Qual?

```
public int getX2() {  
    return x2;  
}  
public void setX2(int x2) {  
    this.x2 = x2;  
}  
public int getY2() {  
    return y2;  
}  
public void setY2(int y2) {  
    this.y2 = y2;  
}  
}
```

Herança – Redefinição de Métodos

- ▶ O método:

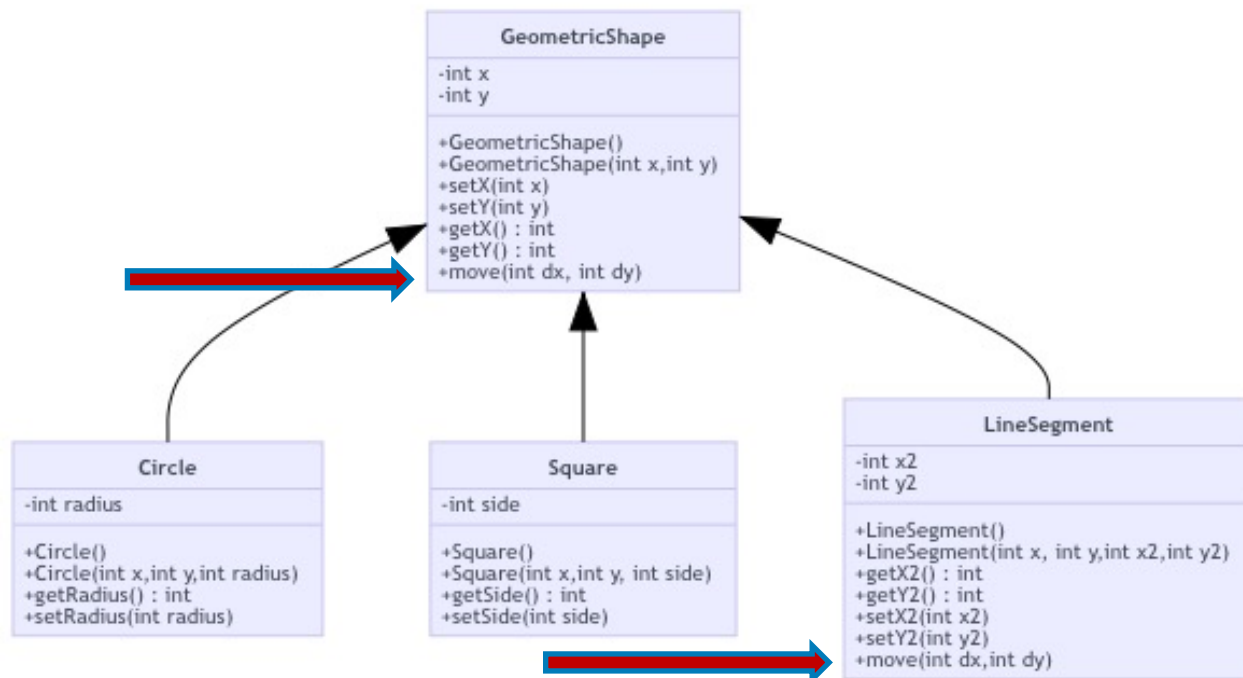
```
public void move( int dx, int dy ) {  
    x += dx; y += dy;  
}
```

- ▶ Este método não funciona corretamente para objetos da classe **LineSegment**.
- ▶ A solução é redefinir este método nesta classe.
- ▶ Para redefinir um método basta defini-lo novamente no corpo da subclasse:



```
public class LineSegment extends GeometricShape {  
    // código omitido  
  
    @Override  
    public void move( int dx, int dy ) {  
        super.move(dx, dy);  
        x2 += dx; y2 += dy;  
    }  
}
```

Exemplo – Formas Geométricas





Considerações Finais

- ▶ Herança de classes

Herança (is-a) – Generalização vs Especialização

- ▶ Uma **superclasse** de outras classes representa a **generalização** dessas classes
 - ▶ A superclasse é mais genérica que as subclasses
 - ▶ Por exemplo um **Vehicle** é uma generalização de **Bike** e **Car**.

Versus ...

- ▶ Uma subclasse de uma dada classe é uma **especialização** dessa classe
 - ▶ As subclasses especializam a sua superclasse
 - ▶ A classe **Dog** é uma especialização da classe **Animal**.

Herança (is-a) – Generalização vs Especialização

- Na prática a herança utiliza-se por **generalização** e por **especialização** de classes

Generalização

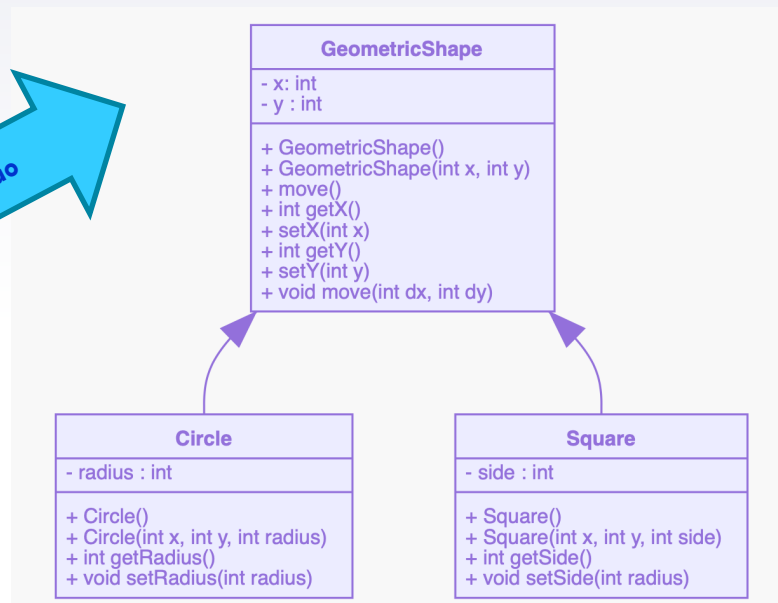
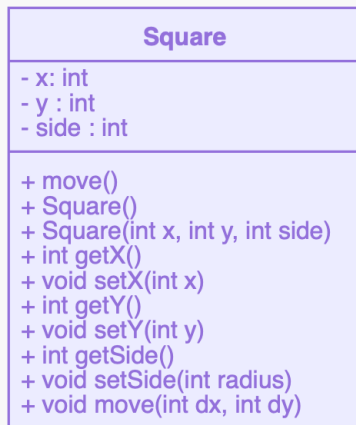
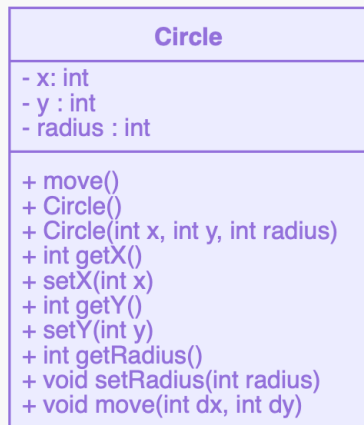
Quando, numa aplicação, se verifica que várias classes partilham um conjunto de atributos e/ou métodos e que a relação de herança pode ser aplicada com a criação duma classe mais genérica que faça sentido dizemos que estamos a usar herança por generalização



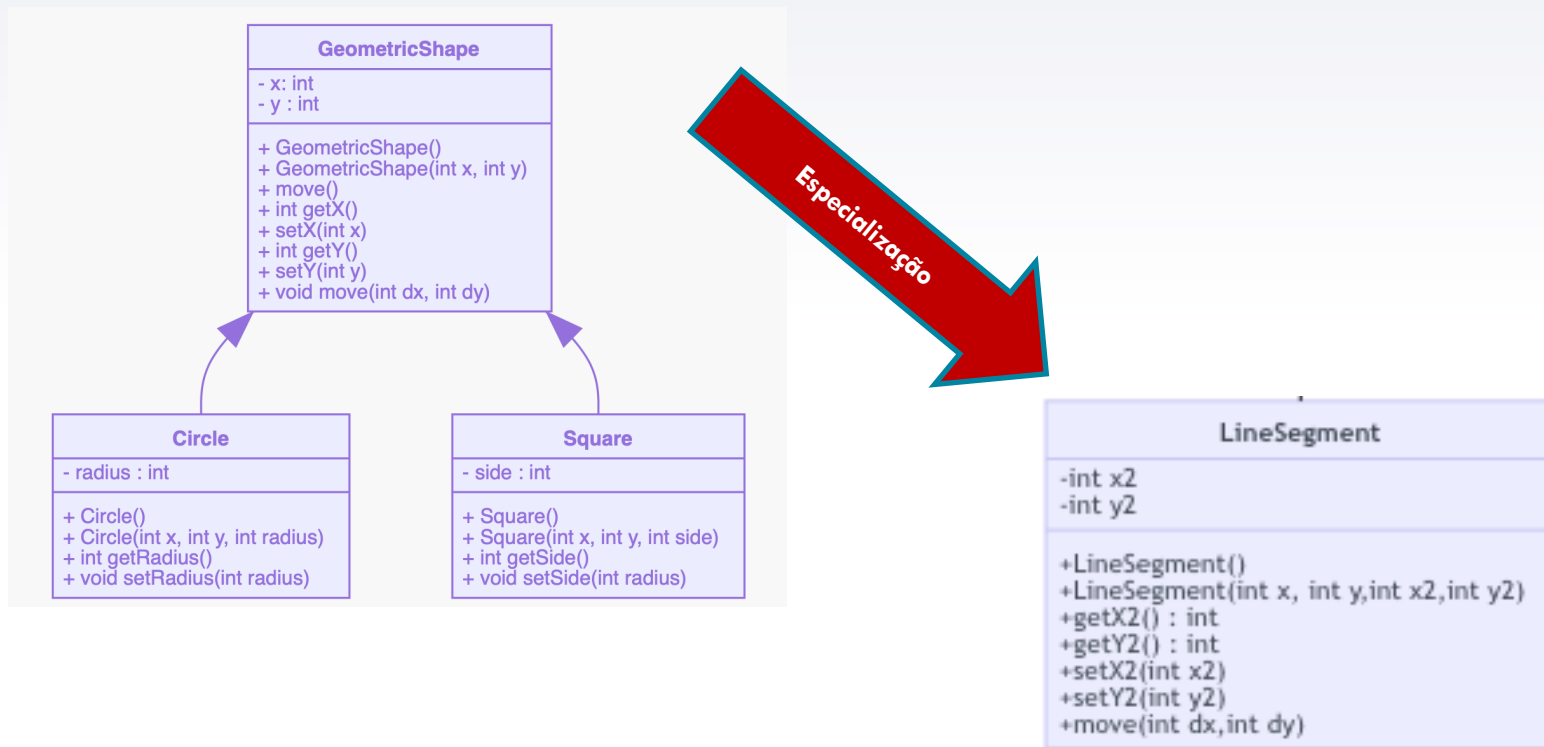
Especialização

Quando já existe classe genérica e se verifica que a classe que se pretende criar pode ser derivada por herança dessa classe e onde essa herança faz sentido, dizemos que estamos a usar a herança por especialização

Herança (is-a) – Generalização vs Especialização




Herança (is-a) – Generalização vs Especialização



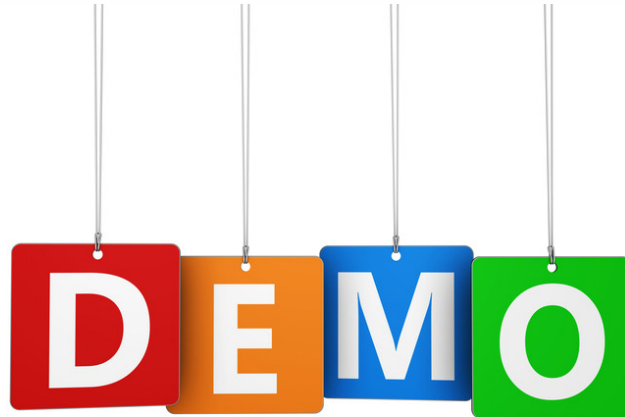
Classe **Object**

- ▶ A classe **Object** é uma classe do Java, que está no topo da hierarquia e da qual todas as outras são subclasses diretas ou indiretas.
 - ▶ Todas as classes são uma especialização de **Object**, são todas um tipo de objeto.
 - ▶ Quando uma classe não deriva de outra o compilador de Java coloca-a a derivar de Object (é acrescentado `extends Object`).
 - ▶ A classe **Object** define um conjunto de métodos que são herdados por todas as classes, entre eles estão os métodos: **toString**, **equals** e **hashCode** utilizados antes.
 - ▶ Por isso quando se coloca um destes métodos numa classe, na prática está-se a redefinir o método herdado, sendo então necessário colocar **@Override** antes da redefinição

 Object
<i>Attributes</i>
<i>Operations</i>
public Object()
public Class getClass()
public int hashCode()
public boolean equals(Object o)
public String toString()
public void notify()
public void notifyAll()
public void wait(long l)
public void wait(long l, int i)
public void wait()

Exemplo – Xadrez (3)

- ▶ Classe **Object**



Bibliografia

- ▶ Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
 - ▶ Capítulo 10

