Paradigmas de Programação

Aula 06

1. Palavras Reservadas
2. Herança
3. Exemplos
4. Super
5. Casting Objects
6. Palavras Reservadas Usadas
7. Links Úteis

Palavras reservadas usadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

```
public class MountainBike{

    private int cadence;
    private int gear;
    private int speed;
    private boolean suspension;

    public MountainBike(int startCadence, int startSpeed,
            int startGear, boolean suspension) {

        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
        this.suspension = suspension;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

   // continua no próximo slide
```

```
    // continuação do slide anterior



    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

    public boolean hasSuspension() {
        return this.suspension;
    }

}
```

MountainBike

Attributes

```
private int cadence
private int gear
private int speed
private boolean suspension
```

Operations

```
public MountainBike( int startCadence, int startSpeed, int startGear, boolean suspension )
public void  setCadence( int newValue )
public void  setGear( int newValue )
public void  applyBrake( int decrement )
public void  speedUp( int increment )
public boolean  hasSuspension( )
```

```
public class RoadBike{

    private int cadence;
    private int gear;
    private int speed;
    private int numberOfBags;

    public RoadBike(int startCadence, int startSpeed,
            int startGear, int numberOfBags) {

        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
        this.numberOfBags = numberOfBags;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

   // continua no próximo slide
```

```
    // continuação do slide anterior



    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

    public int getNumberOfBags() {
        return this.numberOfBags;
    }

}
```

RoadBike

Attributes

```
private int cadence
private int gear
private int speed
private int numberOfBags
```

Operations

```
public RoadBike( int startCadence, int startSpeed, int startGear, int numberOfBags )
public void  setCadence( int newValue )
public void  setGear( int newValue )
public void  applyBrake( int decrement )
public void  speedUp( int increment )
public int  getNumberOfBags(  )
```

RoadBike

Attributes

private int cadence
private int gear
private int speed
private int numberOfBags

Operations

public RoadBike(int startCadence, int startSpee
public void setCadence(int newValue)
public void setGear(int newValue)
public void applyBrake(int decrement)
public void speedUp(int increment)
public int getNumberOfBags( )

MountainBike

Attributes

private int cadence
private int gear
private int speed
private boolean suspension

Operations

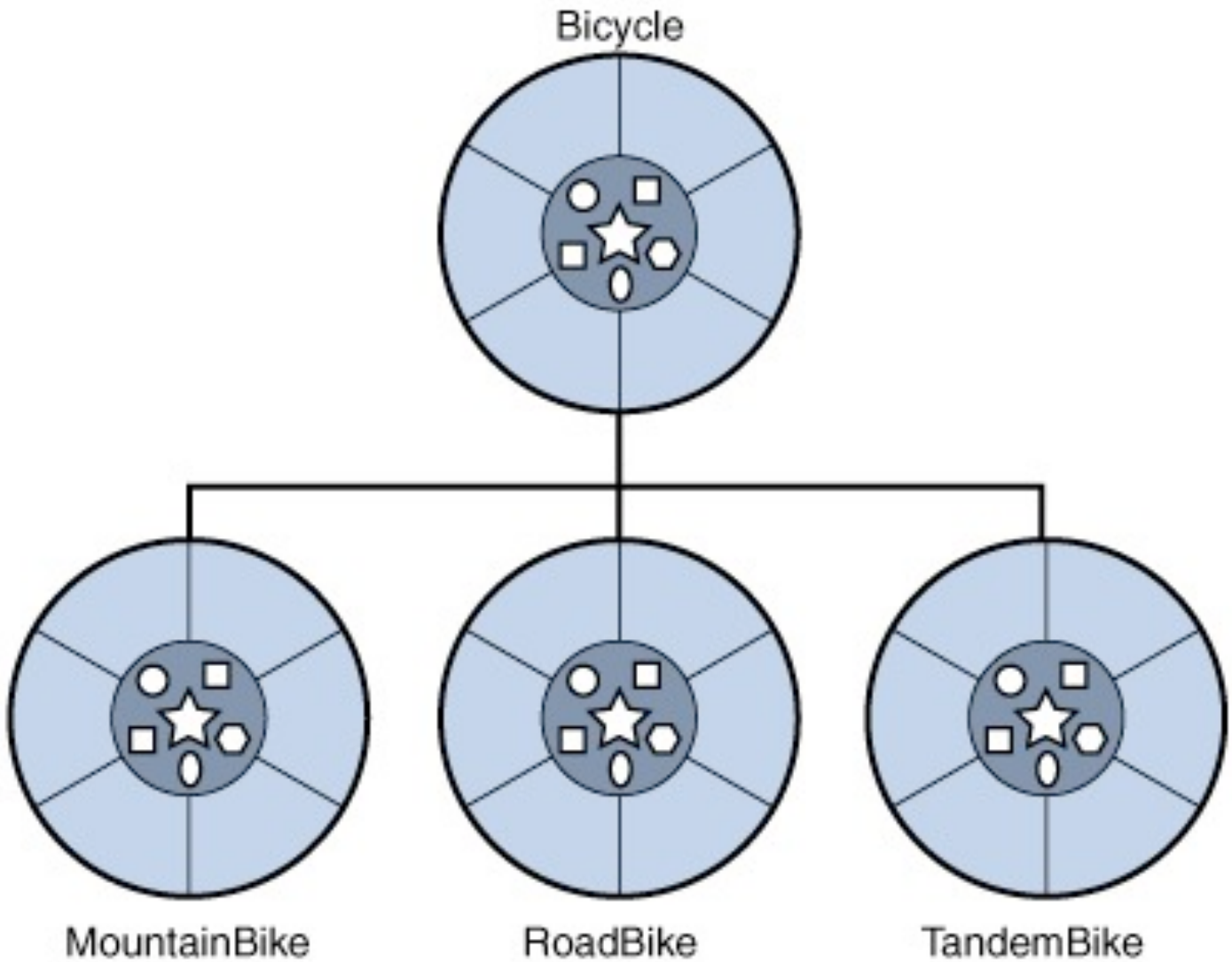
public MountainBike(int startCadence, int s
public void setCadence(int newValue)
public void setGear(int newValue)
public void applyBrake(int decrement)
public void speedUp(int increment)
public boolean hasSuspension( )

O que é a herança?

- Objectos diferentes têm por vezes bastantes semelhanças uns com os outros.
- Por exemplo, bicicletas de montanha, bicicletas de estrada e bicicletas tandem partilham todas as mesmas características (velocidade actual, velocidade do pedalar, mudança).

- No entanto cada uma delas define características adicionais que a diferencia das outras:
  - **bicicletas tandem** têm dois acentos e dois guiadores;
  - **bicicletas de estrada** têm suporte da malas laterais;
  - **bicicletas de montanha** têm suspensão

- A programação orientada a objectos permite que as classes herdem estados e comportamentos que são comuns de outras classes.


Bicycle
MountainBike
RoadBike
TandemBike

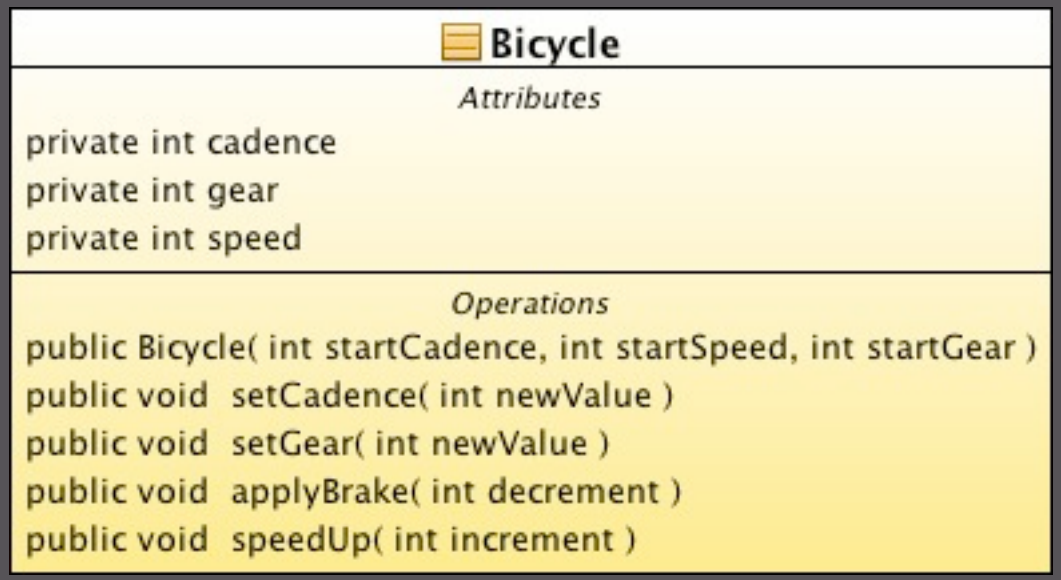
- Na figura anterior, a bicicleta (`Bicycle`) torna-se a super classe de bicicleta de montanha (`MountainBike`), bicicleta de estrada (`RoadBike`) e bicicleta de tandem (`TandemBike`).
- Em java cada classe apenas pode ter uma super classe directa e um número ilimitado de sub-classes

- A sintaxe de criação de uma sub-classe é simples.
- Na declaração da classe usar a palavra reservada **`extends`** seguida do nome da classe que irá herdar. Para o caso da **`MountainBike`**:

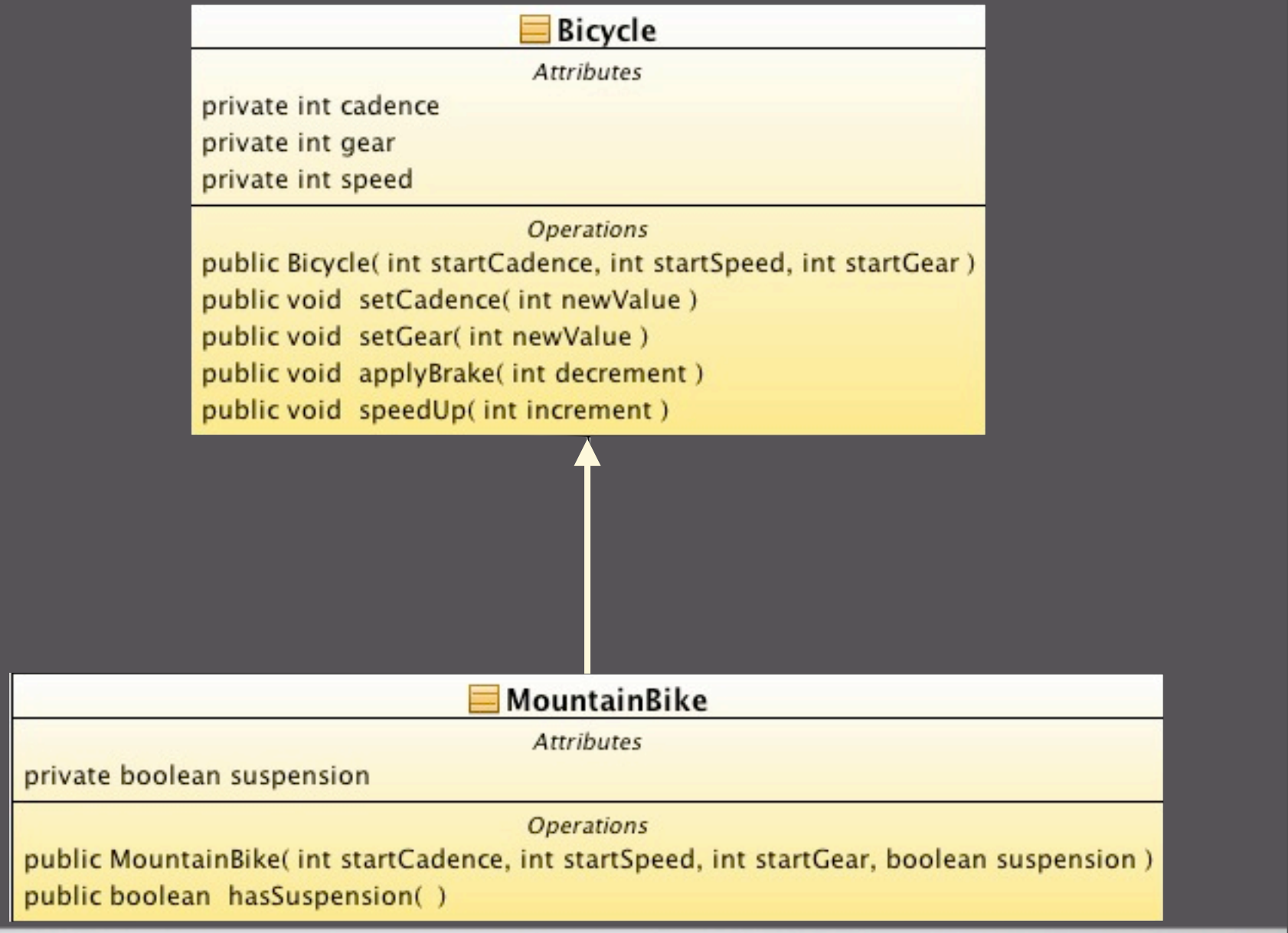
```
class MountainBike extends Bicycle {

     /* new fields and methods defining a
        mountain bike would go here*/

}
```

```java
public class Bicycle {

    // the Bicycle class has three fields
    private int cadence;
    private int gear;
    private int speed;

    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has four methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    // continua no próximo slide
```

```
    // continuação do slide anterior

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

}
```



```java
public class MountainBike extends Bicycle {

    // the MountainBike subclass adds one field
    private boolean suspension;

    // the MountainBike subclass has one constructor
    public MountainBike(boolean suspension, int startCadence,
                        int startSpeed, int startGear) {

        super(startCadence, startSpeed, startGear);
        this.suspension = suspension;
    }

    // the MountainBike subclass adds one method
    public boolean hasSuspension() {
        return this.suspension;
    }

}
```


Bicycle
Attributes
private int cadence
private int gear
private int speed
Operations
public Bicycle(int startCadence, int startSpeed, int startGear)
public void setCadence(int newValue)
public void setGear(int newValue)
public void applyBrake(int decrement)
public void speedUp(int increment)
MountainBike
Attributes
private boolean suspension
Operations
public MountainBike(int startCadence, int startSpeed, int startGear, boolean suspension)
public boolean hasSuspension( )

Super

- A sintaxe para chamar o método construtor de uma superclasse é a seguinte:

```
super();
--or--
super(parameter list);
```

- Com `super()` é chamado o método construtor sem argumentos enquanto que com `super(parameter list)` é chamado o construtor com a assinatura correspondente à lista de parâmetros passado para o `super`
- O `super()` deve ser chamado na primeira linha do método construtor

- Para o exemplo da **`MountainBike`** o construtor de **`bicycle`** é o seguinte:

```
public Bicycle(int startCadence, int startSpeed,
               int startGear) {

      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
  }
```

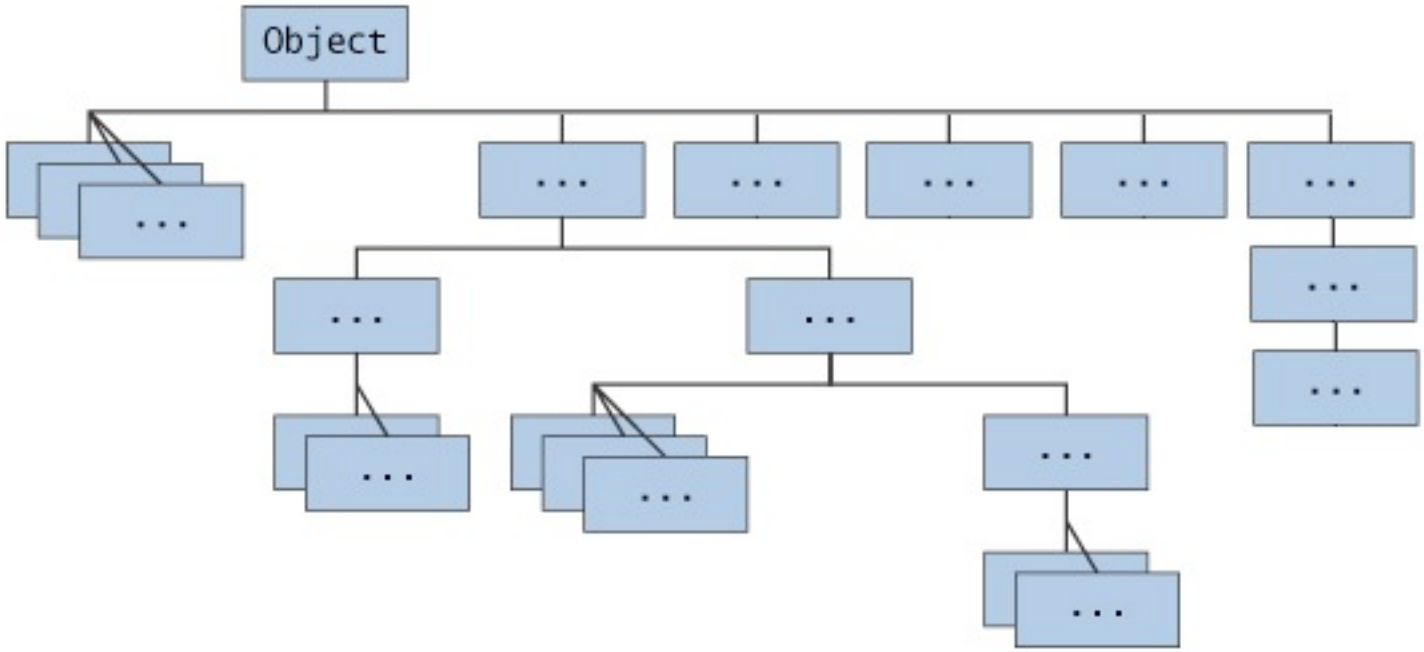
- Repare que na primeira linha do construtor de **`MountainBike`** é feito o super com a respectiva lista de parâmetros:

```
public MountainBike(boolean suspension,
                    int startCadence, int startSpeed,
                    int startGear) {

    super(startCadence, startSpeed, startGear);
    this.suspension = suspension;
}
```

Hierarquia de classes da plataforma java

- A classe `Object` definida no `package java.lang`, define e implementa comportamentos comuns a todas as classes (incluído as criadas por nós).

- Na plataforma java muitas classes derivam directamente de `Object` enquanto que outras derivam dessas classes... formando a hierarquia de classes.


Object
...

Casting Objects

- Temos visto que o tipo de dados de um objecto é da classe com que foi instanciado. Por exemplo:

```
public MountainBike myBike = new MountainBike();
```

- `myBike` é do tipo `MountainBike`

- Como **`MountainBike`** é um descendente de **`Bicycle`** e **`Object`** uma **`MountainBike`** é sempre do tipo **`Bicycle`** e **`Object`**.
- O contrário já não é verdade: **`Bicycle`** pode não ser **`MountainBike`**. O mesmo acontece com o **`Object`**.
- Por exemplo:

```
Object obj = new MountainBike();
```

- **`obj`** é **`Object`** e **`MountainBike`**

- Podemos fazer o seguinte:

```
MountainBike myBike = obj;
```

- Como para o compilador a expressão anterior não é verdadeira temos que lhe dizer explicitamente que o objecto em **`Object`** irá ser do tipo **`MountainBike`**

```
MountainBike myBike = (MountainBike)obj;
```

```
if (obj instanceof MountainBike) {
   MountainBike myBike = (MountainBike)obj;
}
```

- Podemos fazer um teste lógico para determinar se a instância é de um determinado tipo

Palavras reservadas usadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

Links Úteis

- http://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html
- http://docs.oracle.com/javase/tutorial/java/landl/subclasses.html