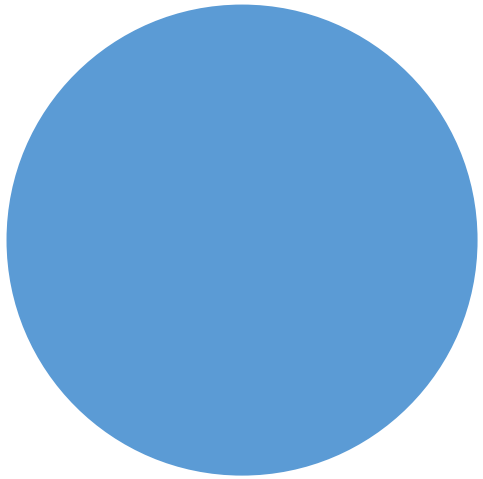
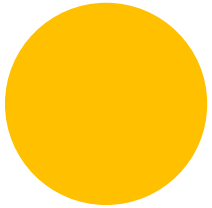



Revisão

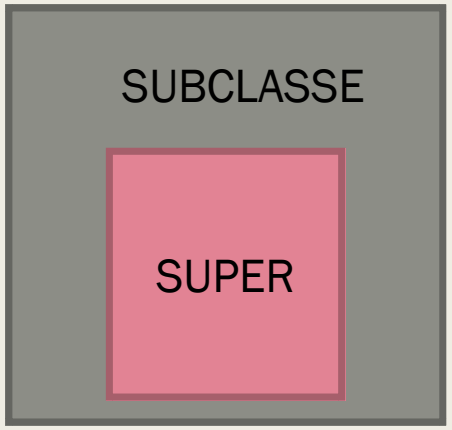
Conceitos Fundamentais
de Programação
Orientada a Objectos

Os 4 conceitos fundamentais

- Herança
- Polimorfismo
- Encapsulamento
- Abstração

Herança

- Processo onde uma classe adquire propriedades de outra.
- Este processo permite que a informação fique mais fácil de organizar e em ordem hierárquica.
- A classe que adquire as propriedades é conhecida como subclasse e a classe cujas propriedades foram herdadas é conhecida como superclasse.



Herança: Exemplo

```
public class HelloWorld{

    public static void main(String []args){

        Sub subclasse = new Sub();
        subclasse.sayHello();
    }
}

class Super {

   public void sayHello() {
       System.out.println("Hello there!");
   }
}

class Sub extends Super {

}
```

Resultado:

Hello there!

Polimorfismo

- Abilidade de uma classe assumir formas diferentes.
- Em programação por objectos o mais comum é um objecto ser tratado como a sua superclasse.
- Qualquer objecto que passe o teste de "IS-A" (é-um) é considerado polimórfico.
- O que determina como um objecto é tratado é a sua referência.

Polimorfismo: Exemplo

```
public class HelloWorld{

    public static void main(String[] args){

        Animal a1 = new Cao();
        Animal a2 = new Gato();
        a1.move();
        a2.move();
    }
}

class Animal {
         public void move() {
                  System.out.println("O animal move-se!");
         }
}
class Cao extends Animal {
}
class Gato extends Animal {
}
```

Resultado:

O animal move-se!
O animal move-se!

Polimorfismo: Overriding

- Capacidade de uma subclasse redefinir um método.

- Tem a vantagem de possibilitar comportamento específico para uma subclasse diferente da superclasse.

Overriding: Exemplo

```
public class HelloWorld{

    public static void main(String[] args){

        Animal a1 = new Cao();
        Animal a2 = new Gato();
        a1.move();
        a2.move();
    }
}
class Animal {
         public void move() {
                  System.out.println("O animal move-se!");
         }
}
class Cao extends Animal {
         public void move() {
                  System.out.println("O cão corre!");
         }

}
class Gato extends Animal {
}
```

Resultado:

O cão corre!
O animal move-se!

Encapsulamento

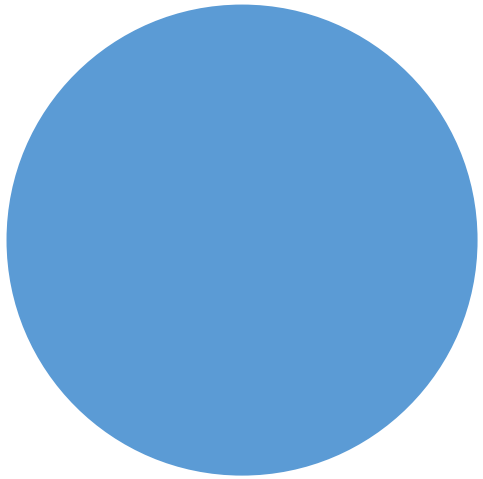
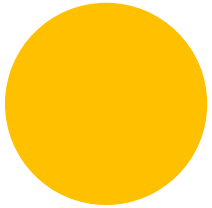
- Encapsulamento é o mecanismo de juntar as variáveis e o código que as controla numa ideia única.
- A ideia é esconder as variáveis e permitir que estas sejam acedidas apenas através de métodos.
- Para isto devemos:
  - *Declarar a variáveis como private.*
  - *Declarar métodos que as alteram como public.*
- Vantagens:
  - *Campos podem ser apenas de leitura ou apenas de escrita.*
  - *Uma classe fica com controlo total sobre como dados são alterados.*

Encapsulamento: Exemplo

```
class Pessoa {
        private int idade;
        private String nome;

        public void fazAnos() {
                idade = idade +1;
        }

        public String getNome() {
                return nome;
        }

}
```

Abstração

- Abstração é o conceito de lidar com ideias em vez de eventos concretos.
- Em programação, trata-se do processo de esconder detalhes de implementação, sendo que apenas a funcionalidade é mostrada ao utilizador.
- O utilizador vai saber o que um objecto faz em vez de saber como é que o faz.

FIM

Exercísios

Exercício 1

- Defina uma classe “Veiculo”. Um veículo deve ter uma certa quantidade de combustível. Deve também ter a capacidade de “buzinar”, representada por um método que imprime a mensagem “PII PII”.

Exercício 2

- Defina o método “abastecer” que recebe um valor de combustível e adiciona-o ao valor de combustível do Veículo. Este método deve retornar o valor total de combustível depois do abastecimento.

Exercício 3

- Defina o método mover que verifica se o Veiculo tem combustível e tenta mover-se. Caso tenho um valor de combustível maior do que zero, deve imprimir a mensagem “VROOM” e diminuir o combustível por 1 unidade. Caso não tenha, deve imprimir uma mensagem “PFFF” a avisar que não se moveu.

Exercício 4

- Crie um novo método “moverUnidade” que recebe um número de unidades de distância e tenta mover o Veiculo esse núemro de vezes. Este método deve chamar o método “mover” o número de vezes que receber como argumento.

Exercício 5

- Crie uma subclasse de Veiculo chamado “VeiculoPesado”. O veículo pesado deve redifinir o método “buzina” para que imprima a mensagem “POO POO”.
- Crie um método main(), crie um Veiculo e um VeiculoPesado referenciados como Veiculo. Chame o método “buzina” em ambos para verificar o comportamento diferente.