**Exame- PPROG**

Os programas Java são formados por **objetos** que interagem entre eles:

* Cada **objeto** é baseado numa **classe**;
* Uma **classe** descreve um conjunto de **objetos** com o mesmo comportamento;
* Cada **classe** define um conjunto específico de **métodos** para usar com os seus **objetos**.

**Dados Privados**:

* Cada **objeto** tem os seus próprios **dados privados** aos quais outros objetos **não** **podem** **aceder** diretamente;
* **Métodos da interface pública** podem aceder a **dados privados**, escondendo detalhes de implementação: **encapsulamento**;
* **Interface Pública**: cada objeto tem um conjunto de métodos disponíveis para uso de outros objetos.

**Especificadores de Acesso**:

* **Classes** (e métodos de interface) são **públicos** (public);
* **Variáveis de instância** são sempre **privadas** (private).

**Métodos Não Estáticos**:

* No caso de métodos não estáticos (**instância**), é necessário **instanciar** um objeto da classe antes que os métodos possam ser invocados;

**Métodos static**:

* Um método estático pertence à classe e não ao objeto de uma classe.

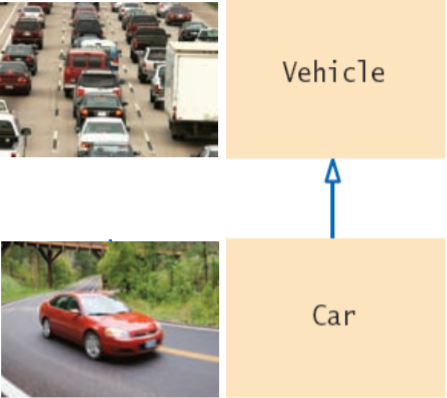
Variáveis de instância **private não podem ser acedidas** a partir de métodos **fora da classe**, só através de métodos de acesso à classe (getters).

Um construtor é um método que inicializa as variáveis de instância de um objeto:

* É chamado automaticamente quando um objeto é criado;
* Tem exatamente o mesmo nome da classe.

**Uma referência não pode apontar nenhum objeto**:

* Não é possível invocar métodos de um objeto através de uma referência null – originará um run-time error
* CashRegister reg = null; System.out.println(reg.getTotal()); // Runtime Error!

Na programação orientada ao objeto, herança é uma relação entre:

* A **subclasse** ‘**herda’** dados (variáveis) e comportamento (métodos) da superclasse;
* Uma **superclasse**: uma classe **mais** **genérica**;
* Uma **subclasse**: uma classe **mais especializada**.
* Do tipo- “**is a**”;
* Ex: Um carro é um veículo.

**Subclasses herdam da superclasse**:

* Todos os métodos públicos que a subclasse não reescreva (override);
* Todas as variáveis de instância;
* A subclasse pode adicionar novas variáveis de instância, adicionar novos métodos e alterar a implementação de métodos herdados.

Uma imagem com texto

Descrição gerada automaticamente

A herança permite que os membros de uma superclasse possam ser reaproveitados na definição de uma subclasse. Esta subclasse tem acesso aos membros públicos e protegidos da superclasse. O polimorfismo, associado à herança, permite que os métodos abstratos definidos numa classe abstrata sejam implementados nas subclasses, podendo estes métodos, nas subclasses, apresentar comportamentos distintos.

**Overloading**: quando dois métodos partilham o mesmo nome mas têm parâmetros diferentes;

**Overriding**: ocorre quando a subclasse define um método com o mesmo nome e exatamente os mesmos parâmetros do método da superclasse.

Uma imagem com texto

Descrição gerada automaticamente**Polimorfismo**: permite manipular objetos que partilhem um conjunto de tarefas, mesmo que as taregas sejam executadas de forma diferente. Facilita a extensibilidade dos programas.

**Polimorfismo** é a capacidade de um objeto poder ser referenciado de várias formas.

O dynamic lookup permite tratar objetos de diferentes classes de maneira uniforme. Esta caraterística é chamada de polimorfismo. Ao pedirmos a vários objetos que executem uma tarefa, cada objeto executa-a à sua maneira.

**Classes abstratas**:

* Se for desejável forçar a reescrever nas subclasses um método de uma classe base, podemos declarar o método como abstract;
* Não é possível instanciar um objeto que tem métodos abstract;
* Só há métodos abstratos em classes abstratas;
* **Se estendermos a classe abstract, é necessário implementar todos os métodos abstratos.**

**Referências Abstratas**:

* Uma classe que pode ser instanciada é denominada classe concreta;
* Uma imagem com texto

  Descrição gerada automaticamenteNão é possível instanciar uma classe que tenha métodos abstract.

**Instanceof**: verifica se um objeto pertence a um tipo particular e devolve um boolean.

**Interface**:

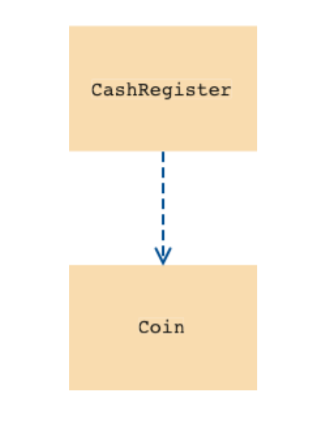
* Lista um conjunto de métodos e assinaturas;
* Todos os métodos são abstratos;
* Todos os métodos são públicos;
* Não pode ter variáveis de instância;
* Não tem métodos estáticos;
* Podem declarar constantes;
* “implements”.

Uma imagem com mesa

Descrição gerada automaticamente

**Coesão**:

* Uma classe deve representar um único conceito;
* A interface pública de uma classe é coesa se todas as suas características estão relacionadas com o conceito que a classe representa.

**Relações entre classes**:

* Uma classe depende de outra se usa objetos dessa classe;
* relação “conhece”.

**Acoplamento**: Se o nível de dependência entre classes é grande, o acoplamento entre classes é elevado.

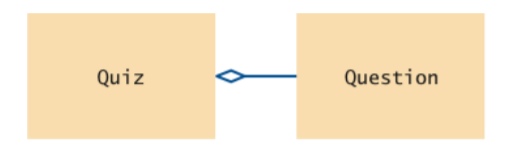
Na programação orientada ao objeto, um objeto está relacionado com outro para **usar funcionalidades e serviços fornecidos por esse objeto**:

* Esta relação entre dois objetos é conhecida como associação;
* Composição e Agregação são formas de associação entre dois objetos.

**Associação:** um objeto está relacionado com outro para usar funcionalidades e serviços fornecidos por esse objeto. Se B deixar de existir, A é eliminado.

**Agregação**:

* Uma classe agrega outra se os seus objetos contêm objetos de outra classe;
* relação “has-a”.
* Se B deixar de existir, A mantém-se. O Quiz mantém-se se deixar de haver uma Question.



**Composição**:

* Referimo-nos à associação entre dois objetos como composição, quando uma classe possui outra classe e a existência desta outra classe não faz sentido, quando o seu “dono” é destruído;
* Relação “part-of”;
* Exemplo: a classe Humano é uma composition de várias partes do corpo incluindo mão, braço e coração;
* Se B deixar de existir, A é eliminado.

**Herança**:

* Herança é uma relação entre uma classe mais genérica (superclass) e uma classe mais especializada (subclass);
* Relação “is-a”.

**Packages**:

* Um package é um conjunto de classes relacionadas;
* Usar packages para estruturar as classes na aplicação;
* A diretiva import permite fazer referência a uma classe de um package, sem utilizar o prefixo do package.

**Um tipo enumerado** é:

* Um tipo especial que permite predefinir um conjunto de valores que poderão ser assumidos por uma variável

**Usar um array**:

* Tamanho nunca muda;
* Necessário armazenar um grande conjunto de valores de um dos tipos primitivos;

**Usar um ArrayList**:

* Restantes situações;
* Se se desconhecer o número de elementos a armazenar.

Existem dois aspetos a ter em conta para lidar com **erros de run-time**:

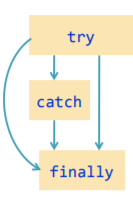
* **Detecção de Erros**- perante uma situação de erro podemos ‘lançar’ (throw) uma exceção;
* **Tratamento de Erros**- tarefa mais complexa – é necessário ‘capturar’ (catch) cada exceção possível e reagir a ela de forma apropriada.

Uma imagem com texto

Descrição gerada automaticamente

**Throw/catch aplica-se a três tipos de exceções**:

* **Error**: Erros internos (ex: OutOfMemoryError)- Raras, não consideradas aqui;
* **Unchecked**: Descendentes de RuntimeException- Devidas a erros no programa (ex: IndexOutOfBoundsException)/ O compilador não verifica se as tratamos;
* **Checked**: Todas as outras exceções- Não são da responsabilidade do programador (ex: IOException)/ O compilador verifica (checks) se as tratamos; se não forem tratadas o programa não compila/ Representadas a escuro no diagrama ao lado.

**Como usar try/catch:**

* Lançar cedo;
* Capturar tarde;
* Não silenciar exceções.

**Serialização**:

* Processo no qual a instância de um objeto é transformada numa sequência de bytes.
* Streams permitem o acesso a sequências de bytes;
* Readers e writers permitem o acesso a sequências de caracteres;
* Usar as classes FileInputStream e FileOutputStream para ler e escrever dados binários;
* O método InputStream.read devolve um inteiro, -1 para indicar final da entrada, ou um byte entre 0 e 255;
* O método OutputStream.write escreve um único byte;
* Os objetos guardados num object stream devem pertencer a classes que implementem a interface Serializable;
* Como referido, se uma superclasse é serializável todas as suas subclasses também o serão.
* Um atributo "transient" é uma atributo que não é serializado durante o processo de serialização (Serialization).

**Exemplo- Serialização:**

**Uma imagem com texto

Descrição gerada automaticamente**

**Uma imagem com texto

Descrição gerada automaticamente**Alternativa: criamos uma classe nova e criamos um objeto da classe que queremos serializar. Depois fazemos o resto.

**Uma imagem com texto, interior, captura de ecrã

Descrição gerada automaticamenteExemplo- Deserialização**

**Uma imagem com texto

Descrição gerada automaticamente**Alternativa:

**Uma imagem com texto

Descrição gerada automaticamenteLinkedList**:

* Permite operações rápidas de inserção e remoção na lista.

**Classes genéricas**:

* Em Java, a programação genérica pode ser obtida por aplicação de herança ou de parâmetros de tipo;
* Uma classe genérica tem um ou mais parâmetros de tipo;
* Parâmetros de tipo podem ser instanciados por classes ou por interfaces;
* Os parâmetros de tipo tornam o código genérico mais robusto e de mais fácil leitura;
* As variáveis de tipo de uma classe genérica aparecem após o nome da classe e são rodeadas por < >;
* Podemos usar parâmetros de tipo para tipos de variáveis de instância genéricas, parâmetros de métodos e valores de retorno de métodos;
* Um método genérico é um método com um parâmetro de tipo;
* Os parâmetros de tipo de um método genérico são colocados entre os modificadores do método e o tipo de retorno do método;
* Na invocação de um método genérico, não é necessário instanciar os parâmetros de tipo;
* Os parâmetros de tipo podem ser restringidos através da definição de limites;
* A máquina virtual “apaga” os parâmetros de tipo, substituindo-os pelos seus limites (se definidos) ou por Objects;
* Não é possível construir objetos ou arrays de um tipo genérico.

Uma imagem com texto

Descrição gerada automaticamente

* Uma imagem com texto

  Descrição gerada automaticamenteMétodos estáticos em interfaces têm de ter corpo;
* Não pode haver final …;

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Resolução:

public class SerializeMain {

public static void main(String[] args) {

Person p = new Person();

p.setName(“Luís”);

p.setAge(20);

Address address=new Address(“Rua 1”, “Cidade 1”, 1, “4990-001”);

p.setAddress(address);

ObjectOutputStream stream = null;

try {

stream = new ObjectOutputStream(new FileOutputStream("info.bin"));

stream.writeObject(p);

stream.close();

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

public class Person implements Serializable {

…

private void writeObject(ObjectOutputStream os) throws IOException, ClassNotFoundException{

try {

os.defaultWriteObject();

os.writeObject(address.getStreet());

os.writeObject(address.getCity());

os.writeInt(address.getDoorNumber());

os.writeObject(address.getZipCode());

} catch (Exception e) {

e.printStackTrace();

}

}

}