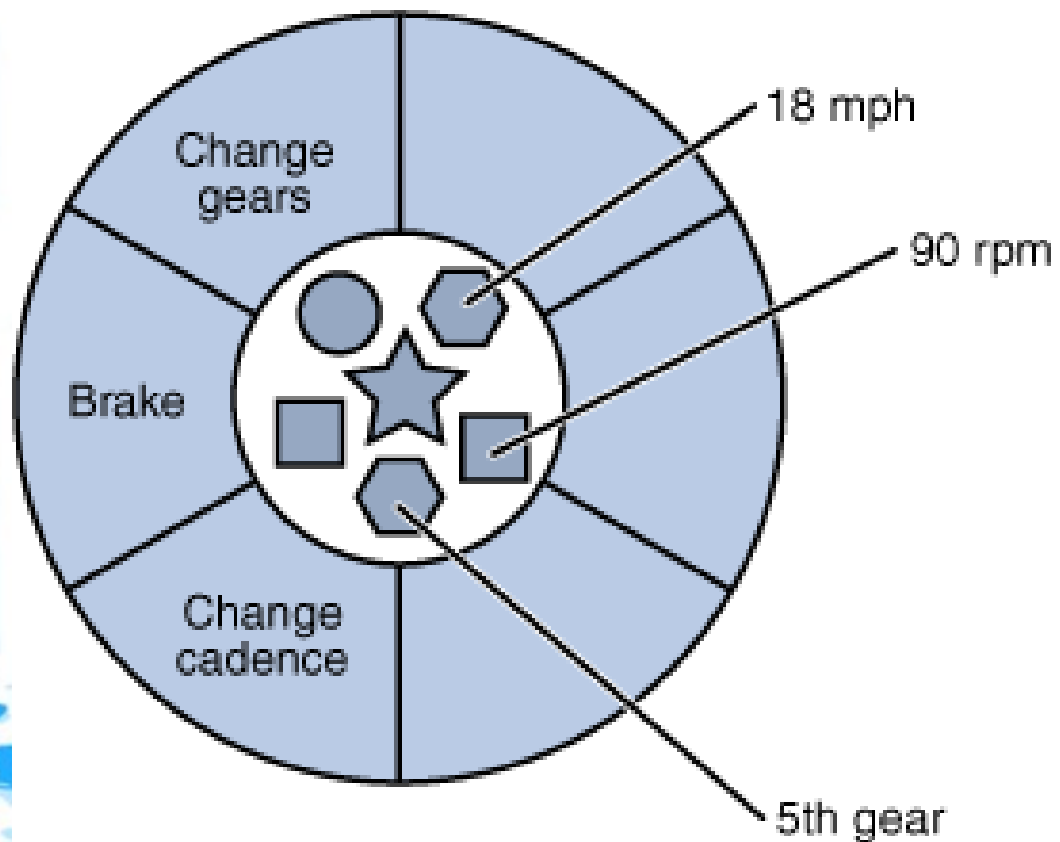


# Revisão

- Objetos são instâncias de Classes, a realização de uma Classe.
- Uma classe é uma especificação de um objeto que define as características (variáveis) e os comportamentos (métodos) e os relacionamentos que os objetos terão entre si.

# Revisão



# Revisão

- Objeto
  - Estado (variáveis)
  - Comportamento (métodos)
  - Identidade (referência)

# Revisão

- Objeto
  - Para ter acesso a alguma variavel ou algum método de um objeto fazemos

objeto.variavel = ...

objeto.metodo(..);

# Revisão

- Stack (pilha) & Heap

# Orientação a Objetos

- Observações importantes:
  - Todas as classes herdam de Object...
  - Os construtores das **super** classes são invocados antes da construção da Classe atual(**this**)

# Orientação a Objetos

- `super()` e `this()`;
  - `super` – classe 'mãe'
  - `this` – classe atual;
  - Exemplo:

```
void mudarX(int x){  
    this.x = x;  
}
```

# Orientação a Objetos

- Sendo super referente a classe 'mãe' então o método `super()` invoca o construtor dessa classe, da mesma forma como `this()` invoca o construtor da classe atual, então é possível de dentro de um construtor invocar outros construtores...

```
public Humano(int idade){  
    super();  
    this(idade,70);  
}
```



# Orientação a Objetos

- Alguns exemplos na prática...

# Orientação a Objetos

- Faça o projeto de um Notebook. Utiliza as superclasses Máquina e Computador e implemente o método ligar() na hierarquia.
  - Obs: Um Computador é composto por uma tela e um teclado que são objetos também...

# Orientação a Objetos

- Faça uma classe Ponto3D que herda de Ponto2D

# Orientação a Objetos

- Métodos
  - Todos os métodos possuem uma assinatura que é composta por
    - Tipo de retorno
    - Nome do método
    - Nro dos argumentos
    - Tipo dos argumentos

# Orientação a Objetos

- Métodos
  - **boolean isBigger(int nro1, int nro2)**
    - boolean
    - isBigger
    - 2 argumentos
    - int, int

# Orientação a Objetos

- Métodos
  - **void comer(String comida)**
    - void
    - comer
    - 1 argumento
    - String

# Orientação a Objetos

- Métodos
  - Sobreescrita
    - A sobreescrita de métodos acontece quando um método é herdado por outra classe e você sobreescreve-o com **a mesma assinatura!**

# Orientação a Objetos

- Métodos
  - Sobrecarga
    - No caso da sobrecarga o método possui o mesmo nome mas possui **assinaturas diferentes!**

# Orientação a Objetos

- Pacotes (introdução)
  - O código pode ser separado em pacotes diferentes, pra melhor organizar a estrutura do sistema...

# Orientação a Objetos

- Pacotes (introdução)

- Exemplo

```
package aula3;  
class Computador {  
    ...  
}
```

# Orientação a Objetos

- Obs: uma vez que vc use pacote é necessário criar um pacote “default” caso contrário as outras Classes não vão enxergar as classes fora de pacotes.

# Orientação a Objetos

- O comando **package** é sempre o primeiro comando de um arquivo... ele vem antes de qualquer coisa!
- Para que a sua Classe dentro de um pacote enxergue outra classe em outro pacote, use o comando **import**

# Orientação a Objetos

- `import nomeDoPacote.NomeDaClasse;`
- `import nomeDoPacote.*;`
  - Importa todas as classes de um pacote

# Orientação a Objetos

- Convenção de Codificação...
  - Como os códigos em java podem ser jogados na internet e pessoas do mundo todo podem abrir e usar eles existe uma convenção de nome de pacotes... Porque?

# Orientação a Objetos

- Convenção de Codificação...

- [cursojava.com.br](http://cursojava.com.br)

- br

- com

- cursojava

- Conteúdo pacote...

- br.com.cursojava.pacote1

- br.com.cursojava.pacote2

# Orientação a Objetos

- Modificadores de Acesso
  - Uma variavel ou um método não devem ser sempre visíveis...
  - Classes são caixa-preta

`pessoa1.idade = 15;` > inseguro!

# Orientação a Objetos

- Pra que eu limite o acesso a variaveis e metodos existem modificadores de acesso!
- Java possui 4 modificadores de acesso...

public

protected

default

private

# Orientação a Objetos

- Modificadores de Acesso

## Public

Uma variável ou método nomeado como público pode ser acessado por qualquer pessoa sem restrição

```
peessoa1.comer();
```

# Orientação a Objetos

- Modificadores de Acesso

**protected**

Uma variavel nomeada como **protected** pode ser acessada por outros objetos que estejam no mesmo **pacote** ou que **herdem** dela

```
peessoa1.comer();
```

# Orientação a Objetos

- Modificadores de Acesso

**default – modificador package** (package-private)

também conhecido como o modificador **padrão**, caso você não coloque modificador algum é esse que será considerado

modificador de **pacote**, será visível dentro do mesmo pacote!

# Orientação a Objetos

- Modificadores de Acesso

**private**

é o modificador com maior restrição, visível apenas na classe.

- Quando uma Classe herda de outros as variáveis private não poderão ser acessadas diretamente...

# Orientação a Objetos

- Modificadores de Acesso

## Classes

Só podem ser “default” ou public ..

## Construtores

- private | protected

## Variáveis

- Qualquer modificador...

## Variáveis locais

- Não tem sentido modificadores...



# Orientação a Objetos

## Modificadores de Acesso

public  
protected  
default  
private

# Orientação a Objetos

- 'static'
  - São variáveis associadas a uma Classe e existem antes de qualquer objeto existir...
  - Variáveis static são comuns a todos os objetos de uma mesma Classe!]

ex. contagem de uma população...

# Orientação a Objetos

- 'static'
  - Podem ser chamados antes da instanciação de qualquer objeto pois estão associados a uma Classe

```
NomeDaClasse.variavelStatic;  
NomeDaClasse.metodoStatic();
```



# Orientação a Objetos

- Exercícios em sala...

# Orientação a Objetos

- Polimorfismo – capacidade de assumir formas diferentes...
  - > veiculo > jipe,moto,aviao,...

# Orientação a Objetos

- Polimorfismo
  - Usar um objeto no lugar de outro...
  - Um objeto intermediário que só fornece as informações necessárias ...
  - ex. Manobrista estacionando Veiculo!

# Orientação a Objetos

- Mas Veiculo é um conceito muito **abstrato**...
  - Tem sentido definir como um veiculo generico freia?
  - Como garantir que as subclasses vao implementar freia() ?

# Orientação a Objetos

- E se não houver subclasses de Veiculo?
  - Um Veiculo não pode ser construído porque ele é apenas um conceito...
    - Mas ele é ótimo como uma abstração, um intermediário, uma interface de comunicação!

# Orientação a Objetos

- Dizemos que Veiculo é uma classe **abstrata** pois possui comportamentos que são **abstratos**, que não podem ser claramente definidos...
- São comportamentos que vão ser definidos nas subclasses

# Orientação a Objetos

- Então se uma classe for abstrata ela não pode ser construída...
  - caso a classe tenha **no mínimo um comportamento abstrato** ela não pode ser construída por completo...

# Orientação a Objetos

- `public abstract class NomeDaClasse { ... }`
  - O modificador `abstract` diz se uma classe é abstrata ou não em java..
  - Classes abstratas **não** podem ser instanciadas, porque não podem ser construídas...

# Orientação a Objetos

- Classes abstratos
  - Uma classe abstrata pura não deveria ter construtor pois não pode ser instanciada, mas caso uma subclasse queira fazer uma referência a um construtor `super(...)`; logo elas podem ter...

# Orientação a Objetos

- Métodos abstratos
  - São métodos que serão implementados (especializados) mas que não estão definidos...
  - 
  - ex. public **abstract** int somar(int a, int b);

# Orientação a Objetos

- Herança Pura vs. Extendida
  - Jegue morde mas Jipe não.. será?

# Orientação a Objetos



# Orientação a Objetos

- Interface é uma classe abstrata “pura”
  - Não tem construtor
  - Só tem métodos abstratos
  - Variáveis são todas constantes...