Recapitulando

- Orientação a objetos: programas organizados em torno da definição de classes, instanciação de objetos e troca de mensagens.
  - Declaração de variáveis de referencia: `Circle c;`
  - Criação/instanciação do objeto: `c = new Circle();`
- Métodos e Construtores:
  - Construtores: (Overload – assinatura)
    - `public Circle() {  .....}`
    - `public Circle(double x, double y, double r) { ... }`
  - Método:
    - `c.area();`
- Packages: grupos de classes
- Import: Inclusão de classes
- Importante: através de regras de uso de packages e imports, Java mantem uma estrutura de nomes livre de colisões.

Variáveis e Métodos de Classes

- Variáveis de Classe: associadas à classe e não aos seus objetos:
  - Definição da classe:

```
public class Circle{
        static int num_circles;
}
```

  - Uso no programa:

```
...
a = Circle.num_circles;
...
```

  - Exemplo: Math.PI;
- Métodos de Classe: associados à classe (declarados como static):
  - Exemplos:
    - public static void main(String args[])
    - public static double sqrt(double v)
    - Math.sqrt
    - Math.max
    - Math.cos

Garbage Collection – Coleta de Lixo

- Java não tem recursos na linguagem para liberar memória.
  - Lembre-se que em c e c++ podíamos usar as instruções release(), dispose();
- O espaço em memória alocado e não utilizado é liberado (limpo) por um processo chamado “Coleta de Lixo”.

Exceções

- Novos termos:
  - try : uma região de código
  - catch : uma condição de exceção
  - finally : uma região de código
- Um método pode throws exceptions
- Exceções são classes e podem ser definidas e estendidas
- Classes do pacote java.lang que são Throwable:
  - java.lang.Exception;
  - java.lang.Error;
- Não utilizar catch para tratar um objeto Error.
- Exceções devem ser tratadas:
  - Uso de try-catch
  - Uso de throws
- try-finally: pode ser usado sem tratar exceções, útil quando desejamos finalizar sessões: fechar arquivos, conexões, etc...)

Classes Abstratas

- Objetivo: implementar alguma funcionalidade, mas deixar a definição de outras funcionalidades para sub-classes.
- Não podemos instanciar objetos de classes abstratas.

Exemplo:

```
abstract public class Exemplo{
        public int a;
        protected int b;
        private int c;
        public String toString() {  return (“Valores” + a +”;” + b +”;” + c); }
        abstract void alerta();
}
```

Estendendo e Implementando Interfaces

- Interfaces declaram features, mas não possuem nenhuma implementação.
- Classes que implementam interfaces precisam implementar todas as features definidas na interface.
- Interfaces devem capturar as features comuns às classes que irão implementa-las.
- O relacionamento tipo “implements” (interface e classe) pode implementar nenhuma ou mais interfaces.
- O relacionamento tipo “extends” (interface e interface) a interface estendida herda apenas as features declaradas na interface que ela estende. Uma interface estende apenas outra interface (não pode estender uma classe).

Estendendo e Implementando Interfaces

- Java permite apenas herança simples entre classes, entretanto, no caso de extensão de interfaces ou de implementação de interfaces, a herança múltipla pode ocorrer.
- Uma classe pode implementar múltiplas interfaces.
- Uma interface pode estender múltiplas interfaces.
- Uma classe que implementa uma interface ela implementa os métodos abstratos declarados em uma interface, sobrepondo estes métodos.

```
interface MinhaInterface{
    void umMetodo(int i);  //um método abstrato
}
class MinhaClasse implements MinhaInterface{
     public void umMetodo(int i) { //implementação}
}
```

Subtipos e Polimorfismo

Dynamic Binding of Methods (Associação Dinâmica de Métodos)

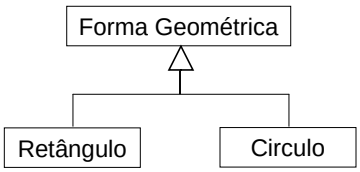
- A chamada a um método é associada a uma implementação específica em tempo de execução, não no momento da compilação.

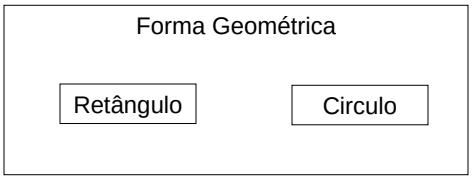
Conceitos Básicos:

- Variáveis: Local de armazenamento com um *tipo* associado. O *tipo* é definido na compilação, estaticamente (*tipo declarado*).
- Objeto: instância de uma *classe*. A *classe* do objeto é definida quando o objeto é *criado,* em tempo de execução.
- Variável de referencia: contém uma referência a um objeto. Pode conter referencia para objetos de diferentes classes, segundo algumas regras. A classe do objeto referenciado nem sempre pode ser determinada na compilação. As vezes somente podemos determinar a classe em tempo de execução.

Subtipos e Polimorfismo

- Subclasse estende a superclasse, herda suas features e adiciona novas.
- Subclasse é uma especialização da superclasse; toda instancia de uma subclasse é uma instancia de sua superclasse. O contrário não é verdade.
- Algumas features estão presentes na subclasse e não na superclasse.
- Cada classe define um tipo. O tipo definido por uma subclasse é um subconjunto do tipo definido pela superclasse.
- O conjunto de todas as instancias de uma subclasse está contido no conjunto de todas as instâncias de uma superclasse.





Subtipos e Polimorfismo

Regra do Subtipo:

O valor de um subtipo pode aparecer sempre que o valor de um supertipo for esperado.

Reescrevendo para o contexto de classes e objetos:

A instancia de uma subclasse pode aparecer sempre que uma instância da superclasse for esperada.

Conversão de Tipos de Referência

A conversão dos tipos de referência depende da relação entre os subtipos:

- *Widening* (Ampliação): A conversão de um subtipo para um de seus supertipo. Sempre permitido. Ocorre muitas vezes de forma implícita.
- *Narrowing* (Redução): A conversão de um supertipo para um de seus subtipos. Requere *cast* explícito e é sempre aceito na compilação, mas pode apresentar problemas na execução.

Diferenças entre tipos primitivos e tipos de referência:

- A conversão entre tipos primitivos resulta na alteração da forma de representação dos valores que estão sendo convertidos
  - a conversão de um inteiro para um double vai implicar na mudança do formato de representação de int para double
- A conversão de um tipo de referência não altera a representação do objeto. Sua identidade e estado permanecem inalterados.

Atribuição Polimórfica

Em linguagens de programação estáticas, como por exemplo o C, em uma atribuição, os dois membros (da esquerda e da direta) precisam ter tipos compatíveis.

```
char c = 's';     //ok
int i = c;    //erro
```

Linguagens de programação orientadas a objetos possuem uma forma mais poderosa de realizar atribuições, as atribuições polimórficas.

O tipo da expressão do lado direito da atribuição precisa ser um subtipo do tipo da variável do lado esquerdo da atribuição.

Atribuição Polimórfica – Exemplos (1)

Definição de Classes:

```
class Estudante { ...}
class EstudanteGinasio extends Estudante {...}
class EstudanteSuperior extends Estudante {...}
```

Instanciação das classes em um programa:

```
Estudante estudante1, estudante2;     //variáveis de ref.
estudante1 = new EstudanteGinasio();  //supertipo = subtipo
estudante2 = new EstudanteSuperior(); //supertipo = subtipo
```

Todos as subclasses possuem os métodos definidos na superclasse.

Atribuições do tipo *Widening* (Ampliação).

Atribuição Polimórfica – Exemplos (2)

Instanciação das classes em um programa:

```
Estudante estudante1, estudante2;     //variáveis de ref.
EstudanteGinasio estudante3;          //variável de ref.
estudante1 = new EstudanteGinasio();  //supertipo = subtipo
estudante2 = new EstudanteSuperior(); //supertipo = subtipo

estudante3 = estudante1;    //erro de compilação
```

Um erro de compilação irá ocorrer, pois o tipo de **`estudante1`** é ***Estudante*** e o tipo da variável de referência **`estudante3`** é ***EstudanteGinásio***, apesar da variável de referência **`estudante1`** estar referenciando uma instância da classe ***EstudanteGinásio***.

Neste caso verificamos que estamos realizando um Narrowing (Redução), pois estamos atribuindo a um subtipo um supertipo. Neste caso, o *cast* é necessário.

```
estudante3 = (EstudanteGinasio) estudante1;
```

Atribuição Polimórfica - *Cast*

A realização de *cast* entre variáveis de referência é sempre permitida pelos compiladores Java. A verificação da validade do *cast* é verificada sempre em tempo de execução. No caso de *cast* inválido, uma *exception* é lançada, a *exception* **`ClassCastException`**.

```
estudante3 = (EstudanteGinasio) estudante1;
```

Neste exemplo, em tempo de execução uma verificação é realizada para conferir se a variável estudante1 contém ou não uma referência para um objeto que é uma instância da classe ***EstudanteGinasio*** ou uma de suas subclasses.

No caso desta verificação não ser bem sucedida, a *exception* **`ClassCastException`** é lançada.

Atribuição Polimórfica – Exemplos (3)

Instanciação das classes em um programa:

```
Estudante estudante1, estudante2;     //variáveis de ref.
EstudanteGinasio estudante3;          //variável de ref.
estudante1 = new EstudanteGinasio();  //supertipo = subtipo
estudante2 = new EstudanteSuperior(); //supertipo = subtipo
estudante3 = (EstudanteGinasio) estudante2;
```

Neste caso, a compilação vai ocorrer sem nenhum incidente, entretanto, na execução teremos problemas. O problema é que a variável estudante2 contém uma referência a um objeto da classe EstudanteSuperior, que não é um subtipo da classe EstudanteGinasio.

Atribuição Polimórfica – Tratando Problemas de *Downcast*

Quando uma atribuição polimórfica em que ocorre *narrowing* é realizada, existem duas formas de verificar se esta atribuição está sendo realizada de forma consistente.

- Utilização do operador **`instanceof`** antes da atribuição

```
if (estudante2 instanceof EstudanteGinasio){
   estudante3 = (EstudanteGinasio) estudante2;
} else {
   //estudante2 não é uma instância de EstudanteGinasio
}
```

- Capturar a *exception* **`ClassCastException`**

```
try{
   estudante3 = (EstudanteGinasio) estudante2;
} catch (ClassCastException e) {
   //estudante2 não é uma instância de EstudanteGinasio
}
```

Atribuição Polimórfica – Downcasting? (1)

```
class Estudante { ...}
class EstudantePos extends Estudante {
   public String getTemaPesquisa(){...};
}

Estudante estudante1 = new EstudantePos();
Estudante1.getTemaPesquisa();

Estudante estudante1 = new EstudantePos();
if (estudante1 instanceof EstudantePos) {
   EstudantePos estudante2 = (EstudantePos) estudante1;
   estudante2.getTemaPesquisa();
}
```

Atribuição Polimórfica – Downcasting?(2)

Porque declarar uma variável da classe Estudante e não EstudantePos?

- `estudante1` é um parâmetro. O objeto referenciado por `estudante1` foi criado em algum outro ponto do programa e pode ser qualquer uma das subclasses da classe Estudante.
- `estudante1` é um elemento recuperado de uma coleção de objetos, que normalmente contém objetos da classe `Object`, a classe mais geral.
- `estudante1` é um objeto criado utilizando o método `clone()`, que retorna um objeto da classe `Object`.

Redefinição de Métodos (1)

Overriding x Overload

- Métodos de diferentes classes que possuem uma relação de herança.
- Métodos possuem mesmo nome, mesma assinatura e mesmo tipo de retorno.

Overriding se refere a introdução de uma instância de um método em uma subclasse que possui o mesmo nome, assinatura e tipo de retorno de um método da superclasse. A implementação do método na subclasse substitui a implementação definida na superclasse.

Redefinição de Métodos – Exemplo (1)

Exemplo de Overload:

```
class ClasseA {
   public void Metodo1()  {...}
   public void Metodo1(int i) { ...}
}

ClasseA a = new ClasseA();
a.Metodo1();
a.Metodo1(10);
```

Redefinição de Métodos – Exemplo (2)

Exemplo Overriding:

```
class ClasseB {
   public void Metodo1()  {...}
}

class ClasseC extends ClasseB{
   public void Metodo1()  {...}
}

ClasseB b = new ClasseB();
ClasseC c = new ClasseC();

b.Metodo1();
c.Metodo1();
```