SCC0504 - Programação Orientada a Objetos

Polimorfismo

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Polimorfismo

Polimorfismo

- A palavra significa “várias formas”
- Habilidade de uma mensagem (método) ser apresentado em mais de uma forma
- É muito importante para a orientação a objetos!
- Definição de 1 interface que possui múltiplas interpretações

Polimorfismo

- Existem vários tipos de polimorfismo. E eles podem ser categorizados em dois tipos, baseados na implementação. São eles os polimorfismos:
  - **Estático** ou **Em tempo de compilação**
  - **Dinâmico** ou **Em tempo de execução**

Estático

- O compilador é capaz de definir o resultado do polimorfismo em tempo de **compilação**
- Pode ser de dois tipos
  - Sobrecarga de função (ou método)
  - Sobrecarga de operador (não tem em Java!)

Dinâmico

- Também conhecido como **Despacho de Método Dinâmico** (**Dynamic Method Dispatch**)
- É feita com Sobrescrita de Métodos!
  - Vimos aula passada :)
- Uma chamada de função para o método sobrescrito é resolvida em tempo de execução

Polimorfismo Estático

Polimorfismo Estático

- Sobrecarga de método
  - Múltiplas funções com mesmo nome mas parâmetros diferentes são chamadas de **sobrecarregadas**
- Métodos podem ser sobrecarregados por uma mudança no número de argumentos e/ou nos tipos dos argumentos

Sobrecarga de Método

- Muito útil para criarmos métodos semelhantes, mas que mudam de funcionamento de acordo com os parâmetros
- Por exemplo métodos matemáticos
  - Operações com inteiros vs doubles
  - Operações com 1, 2, 3 parâmetros...

Sobrecarga de Método - Exemplo 1

```
public class MathTest{
    static int Add(int a, int b){
        return a + b;
    }
    static double Add(double a, double b){
        return a + b;
    }
    static String Add(char a, char b){
        return a + Character.toString(b);
    }
}
```

Sobrecarga de Método - Exemplo 1

```
class Main {
    public static void main(String[] args)
    {
        System.out.println(MathTest.Add(2, 4));

        System.out.println(MathTest.Add(5.5, 6.3));

        System.out.println(MathTest.Add('a', 'b'));
    }
}
```

Sobrecarga de Método - Exemplo 2

```
public class GameCharacter{

    int health;

    public GameCharacter(int health){
        this.health = health;
    }

    void TakeDamage(int damage){
        health-=damage;
    }
...
```

Sobrecarga de Método - Exemplo 2

```
...
    void TakeDamage(int damage, String type){
        if(type.compareTo("Ice") == 0)
            TakeDamage(2*damage);
        else
            TakeDamage(damage);
    }
    void TakeDamage(){
        health = 0;
    }
    public int getHealth(){ return health; }
}
```

Sobrecarga de Método - Exemplo 2

```
public static void main(String[] args){

    GameCharacter character = new GameCharacter(10);

    character.TakeDamage(2);
    System.out.println(character.getHealth());

    character.TakeDamage(1, "Ice");
    System.out.println(character.getHealth());

    character.TakeDamage();
    System.out.println(character.getHealth());
}
```

Polimorfismo Estático

- Sobrecarga de operador
  - Você pode usar o mesmo operador (O sinal de ‘+’, por exemplo) para realizar diferentes ações
- Java usa sobrecarga de operadores somente no ‘+’ e não permite ao programador criar novas sobrecargas
- C++, C# e Python permitem!

Sobrecarga de Operador

- No caso do C++
  - Quase todo operador pode ser sobrecarregado, exceto
    - .
    - ::
    - ?:
    - sizeof
  - Ao menos um dos operandos tem que ser um objeto de uma classe definida pelo usuário

Sobrecarga de Operador - Exemplo [2]

```
class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0) {real = r; imag = i;}
    Complex operator + (Complex const &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print() { cout << real << " + i" << imag << endl; }
};
```

Sobrecarga de Operador - Exemplo [2]

```
int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

➔ Saída: `12 + i9`

Polimorfismo Dinâmico

Polimorfismo Dinâmico

- Sobrescrita de Método
  - Quando uma classe derivada tem uma definição para uma das funções da classe base
- Você pode usar uma variável do tipo base, instanciar ela como um objeto da classe pai ou uma das filhas e chamar o mesmo método pra variável, não importa qual a classe instanciada.

Polimorfismo Dinâmico

```
class Pokemon{
  int hp, atk, def;
  String type;
  void takeDamage(amount, enemyType){
      hp-=amount;
  }
}
```

Fonte: https://wall.alphacoders.com/big.php?i=677583

Polimorfismo Dinâmico

```
class WaterPokemon extends Pokemon{
  type = "water";
  void takeDamage(amount, enemy_type){
    if(enemy_type == "fire"){
      hp = hp - (amount/2);
    }
    else{
      hp = hp - amount;
    }
  }
}
```

Fonte: https://www.pokemon.com/br/pokedex/seaking

Polimorfismo Dinâmico

```
class IcePokemon extends Pokemon{
  type = "ice";
  void takeDamage(amount, enemy_type){
    if(enemy_type == "fire"){
      hp = hp - (amount*2);
    }
    else{
      hp = hp - amount;
    }
  }
}
```

Fonte: https://www.pokemon.com/br/pokedex/glalie

Polimorfismo Dinâmico

```
public static void main(String[] args){
    Pokemon poke1, poke2, poke3;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);
    poke1.takeDamage(4, "fire");
    poke2.takeDamage(4, "fire");
    poke3.takeDamage(4, "fire");
    System.out.println(poke1.getHp());
    System.out.println(poke2.getHp());
    System.out.println(poke3.getHp());
}
```

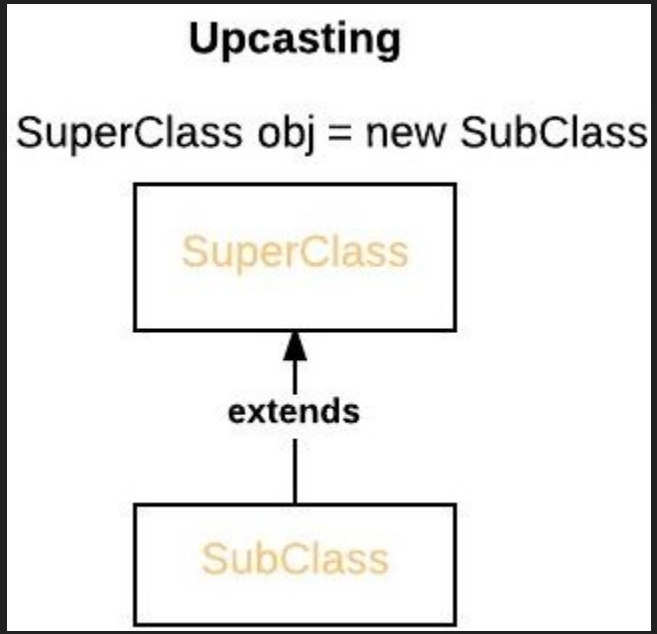
Polimorfismo Dinâmico

```
public static void main(String[] args){

    ArrayList<Pokemon> pokemonList = new ArrayList<>();
    pokemonList.add(new Pokemon(10));
    pokemonList.add(new WaterPokemon(10));
    pokemonList.add(new IcePokemon(10));

    for (Pokemon pokemon: pokemonList)
    {
        pokemon.takeDamage(4, "fire");
        System.out.println(pokemon.getHp());
    }
}
```

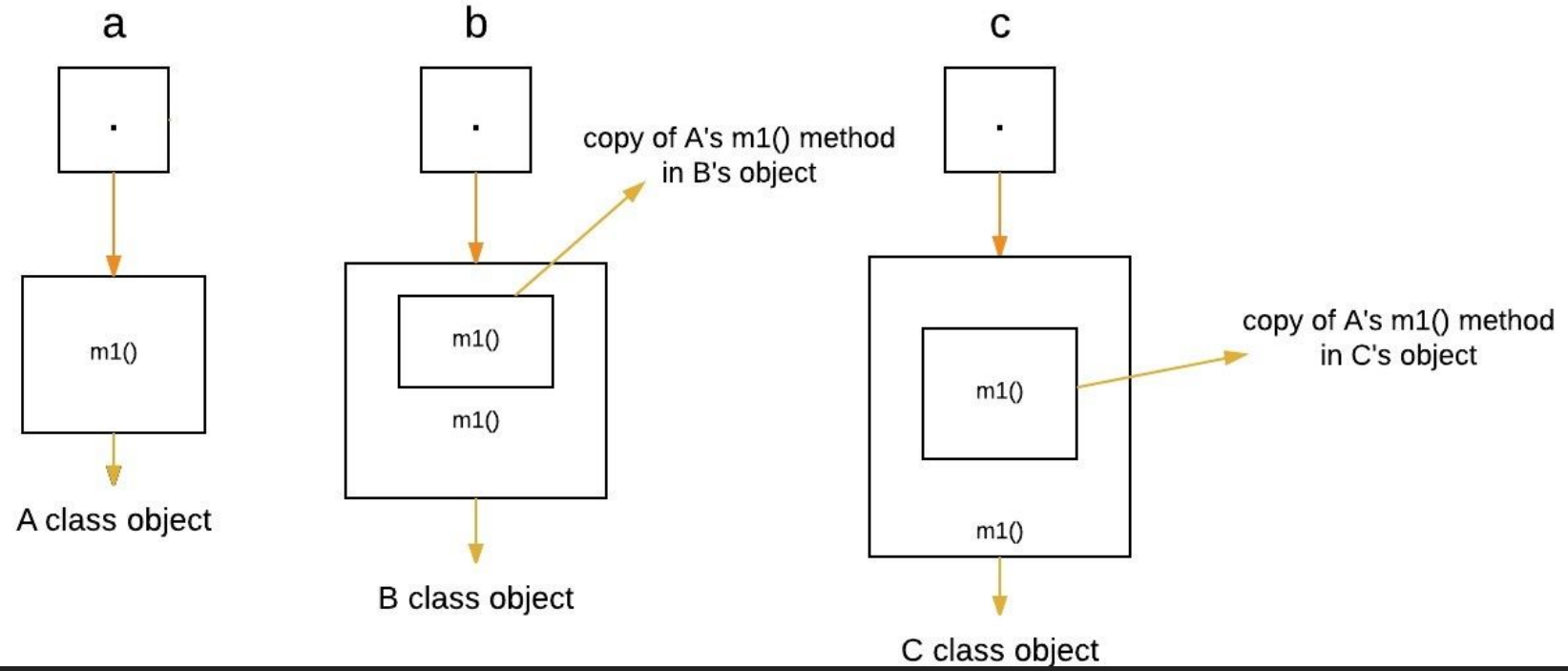
O que acabou de acontecer?

Polimorfismo Dinâmico

- Quando um método sobrescrito é chamado através de uma referência à superclasse, o Java determina qual versão daquele método será executada (classe base ou filha) baseando-se em qual é o tipo do objeto referenciado no momento da chamada
  - Logo, é feito em tempo de execução
  - Também é conhecido como *upcasting*



Fonte: https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/



Fonte: https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/

Polimorfismo Dinâmico

```
public static void main(String[] args){
    Pokemon poke1, poke2, poke3, poke4;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);

    poke4 = poke1;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());

    poke4 = poke2;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());

    poke4 = poke3;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());
}
```

Polimorfismo Dinâmico

- NÃO é possível fazer polimorfismo dinâmico com membros de dados das classes
- Variáveis declaradas nas classes com mesmo nome, mas com valores diferentes atribuídos na **declaração**
- Se “ponteiro” do objeto for do tipo da classe base, mesmo que instanciemos ele como um objeto da subclasse, ele terá o valor das variáveis inicializadas na classe base

Polimorfismo Dinâmico

```
public static void main(String[] args)
{
    Pokemon poke1, poke2, poke3;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);
    System.out.println(poke1.type + " atk: "+poke1.atk);
    System.out.println(poke2.type + " atk: "+poke2.atk);
    System.out.println(poke3.type + " atk: "+poke3.atk);
}
```

Referências

1. https://www.geeksforgeeks.org/polymorphism-in-java/
2. https://www.geeksforgeeks.org/operator-overloading-c/
3. https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/?ref=rp
4. https://beginnersbook.com/2013/03/polymorphism-in-java/
5. https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
6. https://www.programiz.com/java-programming/polymorphism
7. https://www.baeldung.com/java-polymorphism

Extras

- https://www.geeksforgeeks.org/operator-overloading-in-python/
- https://en.wikipedia.org/wiki/Ad_hoc_polymorphism
- https://en.wikipedia.org/wiki/Polymorphism_(computer_science)
- https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/operator-overloading

SCC0504 - Programação Orientada a Objetos

Polimorfismo

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Polimorfismo

Polimorfismo

- A palavra significa “várias formas”
- Habilidade de uma mensagem (método) ser apresentado em mais de uma forma
- É muito importante para a orientação a objetos!
- Definição de 1 interface que possui múltiplas interpretações

Polimorfismo

- Existem vários tipos de polimorfismo. E eles podem ser categorizados em dois tipos, baseados na implementação. São eles os polimorfismos:
  - **Estático** ou **Em tempo de compilação**
  - **Dinâmico** ou **Em tempo de execução**

Estático

- O compilador é capaz de definir o resultado do polimorfismo em tempo de **compilação**
- Pode ser de dois tipos
  - Sobrecarga de função (ou método)
  - Sobrecarga de operador (não tem em Java!)

Dinâmico

- Também conhecido como **Despacho de Método Dinâmico** (**Dynamic Method Dispatch**)
- É feita com Sobrescrita de Métodos!
  - Vimos aula passada :)
- Uma chamada de função para o método sobrescrito é resolvida em tempo de execução

Polimorfismo Estático

Polimorfismo Estático

- Sobrecarga de método
  - Múltiplas funções com mesmo nome mas parâmetros diferentes são chamadas de **sobrecarregadas**
- Métodos podem ser sobrecarregados por uma mudança no número de argumentos e/ou nos tipos dos argumentos

Sobrecarga de Método

- Muito útil para criarmos métodos semelhantes, mas que mudam de funcionamento de acordo com os parâmetros
- Por exemplo métodos matemáticos
  - Operações com inteiros vs doubles
  - Operações com 1, 2, 3 parâmetros...

Sobrecarga de Método - Exemplo 1

```
public class MathTest{
    static int Add(int a, int b){
        return a + b;
    }
    static double Add(double a, double b){
        return a + b;
    }
    static String Add(char a, char b){
        return a + Character.toString(b);
    }
}
```

Sobrecarga de Método - Exemplo 1

```
class Main {
    public static void main(String[] args)
    {
        System.out.println(MathTest.Add(2, 4));

        System.out.println(MathTest.Add(5.5, 6.3));

        System.out.println(MathTest.Add('a', 'b'));
    }
}
```

Sobrecarga de Método - Exemplo 2

```
public class GameCharacter{

    int health;

    public GameCharacter(int health){
        this.health = health;
    }

    void TakeDamage(int damage){
        health-=damage;
    }
...
```

Sobrecarga de Método - Exemplo 2

```
...
    void TakeDamage(int damage, String type){
        if(type.compareTo("Ice") == 0)
            TakeDamage(2*damage);
        else
            TakeDamage(damage);
    }
    void TakeDamage(){
        health = 0;
    }
    public int getHealth(){ return health; }
}
```

Sobrecarga de Método - Exemplo 2

```
public static void main(String[] args){

    GameCharacter character = new GameCharacter(10);

    character.TakeDamage(2);
    System.out.println(character.getHealth());

    character.TakeDamage(1, "Ice");
    System.out.println(character.getHealth());

    character.TakeDamage();
    System.out.println(character.getHealth());
}
```

Polimorfismo Estático

- Sobrecarga de operador
  - Você pode usar o mesmo operador (O sinal de ‘+’, por exemplo) para realizar diferentes ações
- Java usa sobrecarga de operadores somente no ‘+’ e não permite ao programador criar novas sobrecargas
- C++, C# e Python permitem!

Sobrecarga de Operador

- No caso do C++
  - Quase todo operador pode ser sobrecarregado, exceto
    - .
    - ::
    - ?:
    - sizeof
  - Ao menos um dos operandos tem que ser um objeto de uma classe definida pelo usuário

Sobrecarga de Operador - Exemplo [2]

```
class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0) {real = r; imag = i;}
    Complex operator + (Complex const &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print() { cout << real << " + i" << imag << endl; }
};
```

Sobrecarga de Operador - Exemplo [2]

```
int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

➔ Saída: `12 + i9`

Polimorfismo Dinâmico

Polimorfismo Dinâmico

- Sobrescrita de Método
  - Quando uma classe derivada tem uma definição para uma das funções da classe base
- Você pode usar uma variável do tipo base, instanciar ela como um objeto da classe pai ou uma das filhas e chamar o mesmo método pra variável, não importa qual a classe instanciada.

Polimorfismo Dinâmico

```
class Pokemon{
  int hp, atk, def;
  String type;
  void takeDamage(amount, enemyType){
      hp-=amount;
  }
}
```

Fonte: https://wall.alphacoders.com/big.php?i=677583

Polimorfismo Dinâmico

```
class WaterPokemon extends Pokemon{
  type = "water";
  void takeDamage(amount, enemy_type){
    if(enemy_type == "fire"){
      hp = hp - (amount/2);
    }
    else{
      hp = hp - amount;
    }
  }
}
```

Fonte: https://www.pokemon.com/br/pokedex/seaking

Polimorfismo Dinâmico

```
class IcePokemon extends Pokemon{
  type = "ice";
  void takeDamage(amount, enemy_type){
    if(enemy_type == "fire"){
      hp = hp - (amount*2);
    }
    else{
      hp = hp - amount;
    }
  }
}
```

Fonte: https://www.pokemon.com/br/pokedex/glalie

Polimorfismo Dinâmico

```
public static void main(String[] args){
    Pokemon poke1, poke2, poke3;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);
    poke1.takeDamage(4, "fire");
    poke2.takeDamage(4, "fire");
    poke3.takeDamage(4, "fire");
    System.out.println(poke1.getHp());
    System.out.println(poke2.getHp());
    System.out.println(poke3.getHp());
}
```

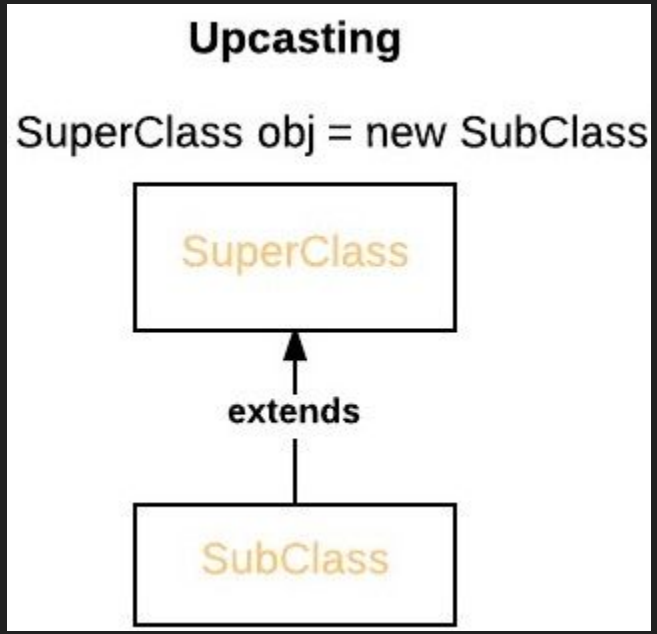
Polimorfismo Dinâmico

```
public static void main(String[] args){

    ArrayList<Pokemon> pokemonList = new ArrayList<>();
    pokemonList.add(new Pokemon(10));
    pokemonList.add(new WaterPokemon(10));
    pokemonList.add(new IcePokemon(10));

    for (Pokemon pokemon: pokemonList)
    {
        pokemon.takeDamage(4, "fire");
        System.out.println(pokemon.getHp());
    }
}
```

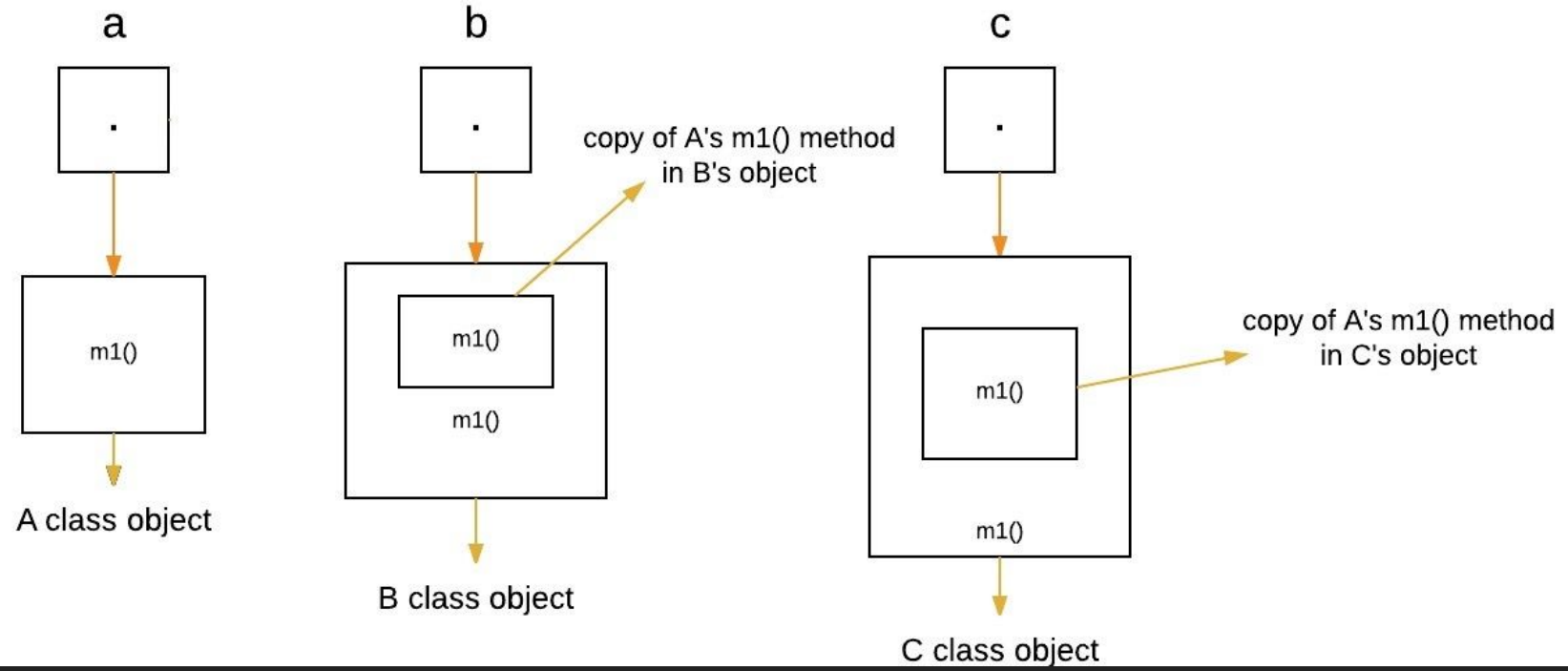
O que acabou de acontecer?

Polimorfismo Dinâmico

- Quando um método sobrescrito é chamado através de uma referência à superclasse, o Java determina qual versão daquele método será executada (classe base ou filha) baseando-se em qual é o tipo do objeto referenciado no momento da chamada
  - Logo, é feito em tempo de execução
  - Também é conhecido como *upcasting*



Fonte: https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/



Fonte: https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/

Polimorfismo Dinâmico

```
public static void main(String[] args){
    Pokemon poke1, poke2, poke3, poke4;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);

    poke4 = poke1;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());

    poke4 = poke2;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());

    poke4 = poke3;
    poke4.takeDamage(4, "fire");
    System.out.println(poke4.getHp());
}
```

Polimorfismo Dinâmico

- NÃO é possível fazer polimorfismo dinâmico com membros de dados das classes
- Variáveis declaradas nas classes com mesmo nome, mas com valores diferentes atribuídos na **declaração**
- Se “ponteiro” do objeto for do tipo da classe base, mesmo que instanciemos ele como um objeto da subclasse, ele terá o valor das variáveis inicializadas na classe base

Polimorfismo Dinâmico

```
public static void main(String[] args)
{
    Pokemon poke1, poke2, poke3;
    poke1 = new Pokemon(10);
    poke2 = new WaterPokemon(10);
    poke3 = new IcePokemon(10);
    System.out.println(poke1.type + " atk: "+poke1.atk);
    System.out.println(poke2.type + " atk: "+poke2.atk);
    System.out.println(poke3.type + " atk: "+poke3.atk);
}
```

Referências

1. https://www.geeksforgeeks.org/polymorphism-in-java/
2. https://www.geeksforgeeks.org/operator-overloading-c/
3. https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/?ref=rp
4. https://beginnersbook.com/2013/03/polymorphism-in-java/
5. https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
6. https://www.programiz.com/java-programming/polymorphism
7. https://www.baeldung.com/java-polymorphism

Extras

- https://www.geeksforgeeks.org/operator-overloading-in-python/
- https://en.wikipedia.org/wiki/Ad_hoc_polymorphism
- https://en.wikipedia.org/wiki/Polymorphism_(computer_science)
- https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/operator-overloading