Fundamentos de Java
7-5: Polimorfismo
Atividades Práticas

Objetivos da Lição:

- Aplicar referências de superclasses aos objetos de subclasse
- Escrever código para sobrepor métodos
- Usar envio do método dinâmico para dar suporte ao polimorfismo
- Criar métodos e classes abstratos
- Reconhecer uma sobreposição do método correto
- Usar o modificador final
- Explicar o propósito e as importâncias da classe do Objeto
- Escrever código de um applet que exibe dois triângulos de cores diferentes
- Descrever referências do objeto

Vocabulário:

Identifique a palavra do vocabulário para cada definição a seguir.

	Um conceito em programação orientada por objetos que permite que as classes tenham muitas formas e se comportem como suas superclasses.
	Implementação de métodos em uma subclasse que têm o mesmo protótipo (os mesmos parâmetros, nome do método e tipo de retorno) que outro método da superclasse.
	Uma palavra-chave em Java usada para limitar subclasses de estender uma classe, substituir métodos ou alterar dados.
	Uma propriedade de uma classe estática que torna a classe incapaz de ser estendida ou os dados incapazes de serem alterados.
	Implementação de um método com o mesmo nome que outro método na mesma classe que tem parâmetros diferentes ou um tipo de retorno diferente.
	O processo pelo qual o Java pode determinar qual método será chamado quando os métodos tiverem sido substituídos.
	Uma palavra-chave em Java que permite que as classes sejam executadas, mas elas não podem ser instanciadas (construídas) e, quando aplicadas a métodos, indicam que os métodos devem ser implementados em todas as subclasses da classe.

Tente/solucione:

1. Qual será o resultado do código a seguir?

```
class A
{
    void callthis() {
        System.out.println("Inside Class A's Method!");
    }
}
class B extends A
{
    void callthis() {
        System.out.println("Inside Class B's Method!");
    }
}

class C extends A
{
    void callthis() {
        System.out.println("Inside Class C's Method!");
    }
}

class DynamicDispatch {
    public static void main(String args[])  {
        A a = new A();
        B b = new B();
        C c = new C();
        A ref;

        ref = b;
        ref.callthis();

        ref = c;
        ref.callthis();

        ref = a;
        ref.callthis();
    }
}
```

2. Qual é a diferença entre uma classe abstract e uma interface? Quando é apropriado usar uma classe abstract ou uma interface?

3. Dadas as informações seguintes, determine se elas resultarão: Sempre compilar, às vezes compilar ou não compilar.

public interface A

public class B implements A

public abstract class C

public class D extends C

public class E extends B

Cada classe foi inicializada, mas não está claro para o que elas foram inicializadas:

A a = new...

B b = new...

C c = new...

D d = new...

E e = new...

Os seguintes métodos estão incluídos:

interface A specifies method void methodA()

class C has the abstract method void methodC()

Código:	Sempre compilar, às vezes compilar ou não compilar?
a = new B();	
d = new C();	
b.methodA();	
e.methodA();	
c = new C();	
(D)c.methodC();	

4. Substitua o método toString() da classe abaixo para produzir os resultados, correspondentes ao resultado fornecido. O método toString() deve imprimir todos os valores de 1 até o número especificado em num e, em seguida, imprimir o valor final usando o método getFactorial fornecido.

Considere que a variável int num é um valor global público:

"Factorial: 10! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800"

```
int getFactorial(){

   int factorial;

      for(i = num; num > 0; i--){

             factorial *= num;

      }

      return factorial;

     }

   public String toString() {

   }
```