Atividade - Classes Wrappers em Java

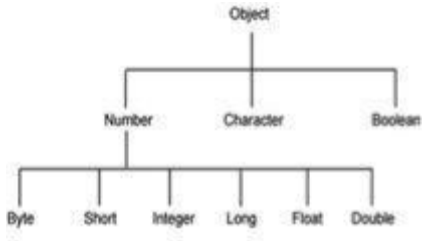

Figura 1. Hierarquia das classes Wrapper.

De acordo com a hierarquia mostrada na Figura 1, cada tipo wrapper numérico em Java são subclasses da superclasse abstrata Number (Java.lang.Number) que consegue acessar todos os métodos *values* que são: **doubleValue(), floatValue(), longValue(), intValue(), shortValue() e byteValue().**

Para fins de estudo, a Tabela 1 mostra todas as classes Wrapper com suas características definidas quando usadas no construtor. Nos construtores Wrappers apenas a classe Character não fornece dois construtores, sendo que as demais aceitam dois construtores.

Tipo primitivo	Classe Wrapper	Argumentos do construtor
Boolean	Boolean	booleanou String
byte	Byte	byte ou String
char	Character	char
Int	Integer	int ou String
float	Float	float, double ou String
double	Double	double ou String
long	Long	long ou String
short	Short	short ou String

Tabela 1. Classes wrapper e argumentos dos construtores

EXERCICIOS

1. O CÓDIGO ABAIXO mostra como funciona na prática a declaração dos construtores mostrados na Tabela 1. Crie um projeto chamado **TrabalhadoComWrappers** e implemente a classe **TestaWrapper** que está no código 1 abaixo.

Código 1 Representação dos construtores de algumas classes Wrapper.

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

                String numInt = "9822";
                //Representação em String do tipo que está sendo criado
                Float fNum1 = new Float("500.50");
                Integer iNum1 = new Integer(numInt);
                Double dNum1 = new Double("512.22");
                //o argumento somente aceita do tipo char, por isso que é usado aspas
                simples
                Character cNum = new Character('a');
                //Criação do tipo primitivo natural
                Float fNum2 = new Float(500.50);
                Integer iNum2 = new Integer(2800);
                Double dNum2 = new Double(512.22);
                System.out.println("Float representadopor string: "+fNum1);
                System.out.println("Float natural: "+fNum2);
                System.out.println("Integer representado por string: "+iNum1);
                System.out.println("Char: "+cNum);

        }
}
```

Nos argumentos wrappers Boolean podem ser usados valores como: true, false ou String. Um ponto de observação é que os valores declarados dentro do construtor não diferencia as letras maiúsculas de minúsculas.

2. O CÓDIGO ABAIXO gera uma amostra do que foi explicado acima. Crie uma classe chamada **TestaWrapperBoolean**, compile, execute e veja o resultado. Seria interessante, tentar adivinhar qual seria a saída dos valores antes de tentar reproduzir para compreender o entendimento.

Código 2 Representação dos construtores de classes WrapperBoolean.

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

        boolean flag1 = true;
        boolean flag2 = false;
        String flag3 = "true";

        Boolean b1 = new Boolean("truE");
        Boolean b2 = new Boolean("TRUE");
        Boolean b3 = new Boolean("TuE");
        Boolean b4 = new Boolean(flag3);

        if(b1){
            System.out.println("B1 é verdadeiro!");
        }

        if(flag1 == b2){
            System.out.println("flag1 == b2: Igual");
        }else{
            System.out.println("flag1 == b2: Diferente");
        }

        System.out.println(flag1 == b1 ? "flag1 == b1: Igual" : "flag1 == b1: Diferente");
        System.out.println(flag1 == b3 ? "flag1 == b3: Igual" : "flag1 == b3: Diferente");
        System.out.println(flag1 == b4 ? "flag1 == b4: Igual" : "flag1 == b4: Diferente");

        Boolean b5 = new Boolean("false");
        Boolean b6 = new Boolean("faLse");
        Boolean b7 = new Boolean("FALSE");
        Boolean b8 = new Boolean("flse");
        System.out.println(flag2 == b5 ? "flag2 == b5: Igual" : "flag2 == b5: Diferente");
        System.out.println(flag2 == b6 ? "flag2 == b6: Igual" : "flag2 == b6: Diferente");
        System.out.println(flag2 == b7 ? "flag2 == b7: Igual" : "flag2 == b7: Diferente");
        System.out.println(flag2 == b8 ? "flag2 == b8: Diferente" : "flag2 == b8: Igual" );
    }
}
```

Saída do código 2:

B1 é verdadeiro!
flag1 == b2: Igual
flag1 == b1: Igual
flag1 == b3: Diferente
flag1 == b4: Igual
flag2 == b5: Igual
flag2 == b6: Igual
flag2 == b7: Igual
flag2 == b8: Diferente

Conversão em wrappers

Para fazer a conversão de tipos primitivos para classes wrappers são usados os seguintes métodos (o *xxx* corresponde ao tipo):

xxxValue() – Esse tipo de método não contém argumentos e é utilizado quando precisa realizar uma conversão do valor de um objeto wrapper para um objeto primitivo.

parseXxx(String s) – Método do tipo primitivo usado para converter um objeto String para um tipo primitivo, sendo que retorna um primitivo nomeado.

valueOf(String s) – Método do tipo wrapper usado para converter um objeto String para um objeto wrapper,ou um tipo primitivo em objeto wrapper, sendo que retorna um objeto wrapper recém criado do tipo que chamou o método.

3. Crie a classe **ConvertWrapper** com forme o código abaixo.
Código 3. Exemplo de conversão xxxValue() - wrapper para tipo primitivo.

```
Public class ConvertWrapper {

        public static void main(String[] args) {
            //cria um objeto wrapper
            Integer velocidade = new Integer(587);

            //converte a variável velocidade para tipo primitivo
            double total = velocidade.doubleValue();
            short total1 = velocidade.shortValue();
            byte total2 = velocidade.byteValue();

            Float pi = new Float(3.14f);
            int valorPi = pi.intValue();

            System.out.println("Valor de PI arredondado: " + valorPi);
            System.out.println("Total: " + total);

        }
  }
```

Saída do código da Listagem 3:
Valor de PI arredondado: 3
Total: 587.0

4. Na classe ConvertWrapper, inclua no método *main* o seguinte trecho:
Código 4. Conversão de uma String para o tipo primitivo.

```
double soma = Double.parseDouble("685.65");
System.out.println("Soma: "+soma);
```

Saída do código da Listagem 4:
Soma: 685.65

5. Inclua o trecho da código 5 no método *main* da classe ConvertWrapper.
código 5. Exemplo do método valueOf com a classe String.

```
Integer idade = new Integer(39);
String idadeString = String.valueOf(idade);
String velocidade = new String("568.55");
Double velocidadeDouble = Double.valueOf(velocidade);
System.out.println("Idade string: "+idadeString);
System.out.println("Velocidade double: "+velocidadeDouble);
```

Comparando variáveis primitivas com wrappers

Na linguagem Java a comparação entre variáveis primitivas, referências e objetos gera certa confusão, pois é necessário tomar cuidados entre o operador de igualdade (==) e o método equals.

Quando existe a comparação apenas com tipos primitivos é utilizado o operador de igualdade (==) e se caso o trabalho for com classes wrappers é invocado o método equals, pois serve para comparação de objetos. Porém, tem algumas restrições com a comparação de classes wrappers e tipos primitivos descritas abaixo:

- Tipos primitivos não conseguem invocar o método **equals**;
- Classes Wrappes não podem ser comparadas com o operador de igualdade (==);

6. Implemente a Classe **ComparandoWrapper** do código 6.

código 6. Comparação com tipos primitivos e classes wrappers

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

                int num1 = 21;
                double num2 = 21.0;
                Double num3 = new Double(282.22);
                Integer num4 = new Integer(232);

                System.out.println(num1 == num2); //true
                System.out.println(num3 == num4); //erro de compilação
                System.out.println(num1 == num3); //false
                System.out.println(num4.equals(num2)); //false
        }
}
```

.