INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

Pacote java.lang

java.lang

- Já **usamos**, por **diversas vezes**, a classe **String**. Vimos o **sistema** de **pacotes** do **Java** e nunca **precisamos** dar um **import** nessa **classe**.

- Isso **ocorre** porque ela está **dentro** do **pacote java.lang**, que é **automaticamente importado** para você. É o **único pacote** com esta **característica**.

- Vamos ver um **pouco** das **principais classes** desse **pacote**.

java.lang.Object

- Quando **declaramos** uma **classe**, essa classe é **obrigada** a **herdar** de **outra**. Porém, criamos diversas **classes** sem **herdar** de **ninguém**:

```
class MinhaClasse {

}
```

- Quando o **Java** não **encontra** a **palavra** chave **extends**, ele **considera** que você está **herdando** da classe **Object**, que está no **pacote** **java.lang** :

```
class MinhaClasse extends Object {

}
```

java.lang.Object

- **Todas as classes, sem exceção, herdam de Object**, seja **direta** ou **indiretamente**, pois ela é a **mãe**, **vó**, **bisavó**, etc de qualquer **classe**.

- Podemos **também afirmar** que qualquer **objeto** em **Java** é um **Object**, podendo ser **referenciado** como tal. Então, **qualquer objeto** possui todos os **métodos** declarados na classe **Object**

Casting de referências

- **Referir** a qualquer **objeto** como **Object** nos traz muitas **vantagens**. Podemos **criar** um **método** que **recebe** um **Object** como **argumento** e podemos **armazenar** qualquer objeto:

```
public class GuardadorDeObjetos {
    private Object[] arrayDeObjetos = new Object[100];
    private int posicao = 0;

    public void adicionaObjeto(Object object) {
        this.arrayDeObjetos[this.posicao] = object;
        this.posicao++;
    }

    public Object pegaObjeto(int indice) {
        return this.arrayDeObjetos[indice];
    }
}
```

- Mas, e no **momento** que **retirarmos** uma **referência** a esse **objeto**, como vamos **acessar** os **métodos** e **atributos** desse **objeto**?

Casting de referências

- Se estamos **referenciando**-o como **Object**, não podemos **acessá-lo** como sendo **Conta**.

```
GuardadorDeObjetos guardador = new GuardadorDeObjetos();
Conta conta = new Conta();
guardador.adicionaObjeto(conta);

// ...

Object object = guardador.pegaObjeto(0); // pega a conta referenciado como objeto
object.getSaldo(); // classe Object nao tem método getSaldo! não compila!
```

- Podemos então **atribuir** essas **referência** de **Object** para **Conta** para depois **invocar** o **getSaldo()**?

```
Conta contaResgatada = object;
```

- Temos certeza de que esse **Object** se **refere** a uma **Conta**. Mas o **compilador Java** não tem **garantias** sobre isso! Essa linha acima **não compila**, pois nem todo **Object** é uma **Conta**.

Casting de referências

- Para **realizar** essa **atribuição**, devemos "**avisar**" o **compilador Java** que realmente queremos fazer **através** do **casting de referências**, **parecido** com de tipos **primitivos**:

```
Conta contaResgatada = (Conta) object;
```

- O código **compila**, mas **roda**? **Sim**, pois em tempo de **execução** a **JVM** verificará se essa **referência** realmente é **para** um **objeto** de tipo **Conta**. Se não estivesse, uma **exceção** do tipo **ClassCastException** seria lançada.

Object: equals e toString

- **Object** tem dois **métodos interessantes**. O **primeiro** método é o **toString**. As classes podem **reescrever** esse **método** para mostrar uma **mensagem**, uma **String**, que o represente.

```
Conta c = new Conta();
System.out.println(c.toString());
```

O método `toString` do `Object` retorna o nome da classe `@` um número de identidade:

```
Conta@34f5d74a
```

- Mas isso **não** é interessante para **nós**. Então podemos **reescreve**-lo:

```
class Conta {
    private double saldo;
    // outros atributos...

    public Conta(double saldo) {
        this.saldo = saldo;
    }

    public String toString() {
        return "Uma conta com valor: " + this.saldo;
    }
}
```

Object: equals e toString

- **Chamando** o **toString**:

```
Conta c = new Conta(100);
System.out.println(c.toString()); //imprime: Uma conta com valor: 100.
```

- Se for **apenas** para **jogar** na **tela**, você nem **precisa** **chamar** o **toString**! Ele já é **chamado** para você:

```
Conta c = new Conta(100);
System.out.println(c); // O toString é chamado pela classe PrintStream
```

Object: equals e toString

- O outro **método importante** é o **equals**. Quando **comparamos** duas **variáveis referência** no **Java**, o **==** verifica se as **duas** referem-se ao mesmo **objeto**:

```
Conta c1 = new Conta(100);
Conta c2 = new Conta(100);
if (c1 != c2) {
    System.out.println("objetos referenciados são diferentes!");
}
```

- Mas, e se fosse **preciso comparar** os **atributos**? O **Java** por si só **não faz isso**, mas **existe** um **método** na classe **Object** que pode ser **reescrito** para **criarmos** esse critério de **comparação**. Esse método é o **equals**.

Object: equals e toString

- O **equals** recebe um **Object** como **argumento** e **verifica** se ele mesmo é **igual** ao **Object recebido** para retornar um **boolean**. Se você não **reescrever** esse **método**, o comportamento **herdado** é fazer um **==** com o **objeto recebido** como argumento.

```
public class Conta {
    private double saldo;
    // outros atributos...

    public Conta(double saldo) {
        this.saldo = saldo;
    }

    public boolean equals(Object object) {
        Conta outraConta = (Conta) object;
        if (this.saldo == outraConta.saldo) {
            return true;
        }
        return false;
    }

    public String toString() {
        return "Uma conta com valor: " + this.saldo;
    }
}
```

Object: equals e toString

- Pelo **contrato definido** pela classe **Object devemos** retornar **false** no caso do objeto **passado** não ser de tipo **compatível** com a sua **classe**.

```
public boolean equals(Object object) {

    if (!(object instanceof Conta))
        return false;
    Conta outraConta = (Conta) object;
    return this.saldo == outraConta.saldo;
}
```

Classes wrappers (box)

- Como **transformar** um **número** em **String** e **vice-versa?**

- O jeito mais **simples** de transformar um **número** em **String** é **concatená-lo** da seguinte maneira:

```
int i = 100;
String s = "" + i;
System.out.println(s);

double d = 1.2;
String s2 = "" + d;
System.out.println(s2);
```

Classes wrappers (box)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Para **transformar** uma **String** em **número**, utilizamos as **classes** de **ajuda;**

```
String s = "101";
int i = Integer.parseInt(s);
```

- As classes **Double**, **Short**, **Long**, **Float** etc contêm o mesmo tipo de **método**, como **parseDouble** e **parseFloat** que retornam um **double** e **float** respectivamente.

- Essas **classes** também são muito **utilizadas** para fazer o **wrapping** (embrulho) de tipos **primitivos** como **objetos;**

Classes wrappers (box)

- **Imagine** que **precisamos passar** como **argumento** um **inteiro** para o nosso **guardador** de **objetos**. Um **inteiro** não é um **Object**, como fazer?

```
int i = 5;
Integer x = new Integer(i);
guardador.adiciona(x);
```

E, dado um `Integer`, podemos pegar o `int` que está dentro dele (desembrulhá-lo):

```
int i = 5;
Integer x = new Integer(i);
int numeroDeVolta = x.intValue();
```

Classes wrappers (box)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Esse processo de **wrapping** e **unwrapping** é **entediante**. O **Java 5.0** em diante traz um recurso chamado de **autoboxing**, que faz isso **sozinho** para você:

```
Integer x = 5;
int y = x;
```

- É importante **ressaltar** que isso **não quer dizer** que tipos **primitivos** e **referências** sejam do mesmo **tipo**, isso é **simplesmente** um "**açúcar sintático**" (*syntax sugar*) para **facilitar** a **codificação**.

java.lang.Math

- Na classe Math, existe uma série de métodos estáticos que fazem operações com números como, por exemplo, arredondar(round), tirar o valor absoluto (abs), tirar a raiz(sqrt), calcular o seno(sin) e outros.

```
double d = 4.6;
long i = Math.round(d);

int x = -4;
int y = Math.abs(x);
```

- Consulte a documentação para ver a grande quantidade de métodos diferentes.

Exercício