JAVA BÁSICO

LABORATÓRIO

1. Abra o **Eclipse** e selecione para “File > New > Java Project”; em **Project Name** coloque “Lab.Heranca” e então confirme.

2. Busque a **pasta** com o nome “src” no lado esquerdo no **Package Explorer**

   (a) Clique com o botão direito na pasta “src” e selecione “New > Class”

   (b) No campo nome digite “Produto” e confirme pressionando **Finish**

3. Caso o novo código não tenha sido aberto automaticamente, busque no **Package Explorer** dentro da pasta “src” o arquivo de classe recém criada “Produto.java”

4. Para a classe produto adicione os seguintes elementos:

   (a) um **int** com nome “id”

   (b) um **Date** com nome “dataCriado” (java.util.Date)

   (c) um Construtor padrão

      i. Inicialize “id” para zero

      ii. Inicialize “dataCriado” com seu construtor padrão

   (d) um Construtor recebendo um **int** para o “id” e um **Date** para o “dataCriado”

5. Crie os gets e sets (públicos) para “id” e “datacriado”

(a) Veja os exemplos das aulas anteriores

6. Compare seu código com o mostrado abaixo

```
import java.util.Date;

public class Produto {

	public Produto()
	{
		this.dataCriado = new Date();
		this.id = 0;
	}

	public Produto(int id, Date dataCriacao)
	{
		this.dataCriado = dataCriacao;
		this.id = 0;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public Date getDataCriado() {
		return dataCriado;
	}

	public void setDataCriado(Date dataCriado) {
		this.dataCriado = dataCriado;
	}

	Date dataCriado;
	int  id;
}
```

7. Agora iremos criar outra classe que herda desta, essa classe se chamará **ProdutoEletronico**

8. Busque a **pasta** com o nome "src" no lado esquerdo no **Package Explorer**

   (a) Clique com o botão direito na pasta "src" e selecione "New > Class"

9. No campo nome digite "**ProdutoEletronico"** e confirme pressionando

10. Agora adicione na classe **ProdutoEletronico** o seguinte comando para herdar da classe **Produto**

```
public class ProdutoEletronico extends Produto {

}
```

11. Adicione os seguintes elementos na classe **ProdutoEletronico**

   (a) Um atributo de nome **serialKey** do tipo **String**

   (b) Get/Set públicos para o atributo **serialKey**

   (c) Um construtor padrão

12. A classe deve estar no formato mostrado abaixo

```
public class ProdutoEletronico extends Produto {


	public ProdutoEletronico()
	{

	}


	public String getSerialKey() {
		return serialKey;
	}

	public void setSerialKey(String serialKey) {
		this.serialKey = serialKey;
	}

	public String serialKey;

}
```

13. Agora adicione um construtor novo que irá chamar o construtor da classe herdada, para isso utilize a sintaxe abaixo

```
	public ProdutoEletronico(int id, String serial)
	{
		super(id, new Date());
		this.serialKey = serial;
	}
```

14. O comando **super** invoca o construtor da classe parente (nesse caso **Produto**)

15. O próximo passo agora é testar a aplicação, para isso crie um método main na classe **ProdutoEletronico** contendo o código mostrado abaixo

```
	public static void main(String[] args)
	{
		ProdutoEletronico pe = new ProdutoEletronico(3, "0349420");
		System.out.println(pe.getSerialKey());
		System.out.println(pe.getDataCriado());
		System.out.println(pe.getId());

		Produto p = new Produto(2, new Date());

		/* Gera erro de compilação */
		// System.out.println(p.getSerialKey());
		System.out.println(p.getDataCriado());
		System.out.println(p.getId());


	}
```

16. Caso a loja decida que todos os produtos eletrônicos devem retornar um **Id** negativo podemos sobrescrever o método de modo a obter o resultado necessário

```
17.         @Override
18.         public int getId() {
19.               return -super.getId();
20.         }
```

NOTA: Perceba uma outra vantagem de utilizar métodos ao invés de atributos: podemos sobrescrever certos comportamentos!

NOTA: Mais a frente falaremos sobre o significado do `@Override` e de outros comandos similares

NOTA: Muitas especialistas não recomendam utilizar herança como reaproveitador de código. Entretanto, na prática habitual (mesmo no framework java e em outras línguagens) isso é muito comum

21.

---

22. Outra funcionalidade possível é impedir que a classe **Produto** seja instanciada; ou seja, um **Produto** so poderá existir se for criado a partir de uma classe herdada, no caso **ProdutoEletronico**

```
public abstract class Produto
```

23. Também podemos obrigar que a classe que herda de **Produto** implemente um método específico, por exemplo:

```
public abstract class Produto {

	public abstract String Categoria();

	// Restante do código
	// <snip>

}
```

24. **NOTA:** Uma classe so poderá ter métodos abstratos se ela for abstrata

25. Uma das maiores vantagens em relação a utilização de herança e polimorfismo é a possibilidade de invocar métodos da classe derivada a partir de uma referência a classe base

```
	public static void main(String[] args)
	{
		Produto p = new ProdutoEletronico();

		// So tem acesso ao métodos de Produto
		System.out.println(p.getId());
	}
```