Reflexão

- O **javax.reflection** é um pacote do Java que permite criar chamadas em tempo de execução, sem precisar conhecer as classes e objetos envolvidos quando escrevemos nosso código (tempo de compilação);
- Esse dinamismo é necessário para resolvermos determinadas tarefas que nosso programa só descobre serem necessárias ao receber dados, em tempo de execução.
- Algumas possibilidades:
  - Listar todos os atributos de uma classe e pegar seus valores em um objeto;
  - Instanciar classes cujo nome só vamos conhecer em tempo de execução;
  - Invocar métodos dinamicamente baseado no nome do método como String;
  - Descobrir se determinados pedaços do código têm annotations.

Reflexão

- O ponto de partida de reflection é a classe **Class**. Esta, é uma classe da própria API do Java onde cada instância representa uma classe do modelo presente;
- Através da **Class** conseguimos obter informações sobre qualquer classe do sistema, como seus atributos, métodos, construtores, etc.

```
Negocio n = new Negocio(); // chamamos o getClass de Object
Class<Negocio> classe = n.getClass();

// outra forma
Class<Negocio> classe = Negocio.class;
```


Reflexão

```
// outra forma
Class classe = Class.forName("Negocio");
```

- Esta é a forma mais versátil, pois permite carregar qualquer classe a partir de uma String;
- Contudo, o nome da classe passada por parâmetro já deve estar previamente carregada no sistema.
- Podemos fazer a carga de classes também em runtime?

```
URLClassLoader child = new URLClassLoader (myJar.toURL(), this.getClass().getClassLoader());
Class classToLoad = Class.forName ("com.MyClass", true, child);
Method method = classToLoad.getDeclaredMethod ("myMethod");
Object instance = classToLoad.newInstance ();
Object result = method.invoke (instance);
```


Reflexão

- Por fim, como criar código em runtime:

```
Class cl = TestePOO.class;
for (Field f : cl.getDeclaredFields()) {
  f.get(instancia);
  f.set(instancia, null);
}
for (Method m : cl.getDeclaredMethods()) {
  m.invoke("p1", "p2");
}
```

- Métodos e campos também podem ser encontrados por nome:

```
Field c1 = cl.getDeclaredField("campo1");
c1.set(instancia, "valor");

Method m1 = cl.getDeclaredMethod("metodo", String.class);
m1.invoke(instancia, "param");
```


Reflexão

- Exercício 6:
  - Desenvolva classes para testar o código desenvolvido no Exercício 2 utilizando reflexão. Deverá carregar as classes principais e imprimir nomes de todos os campos e métodos;
  - Além disso deverá invocar métodos do Banco para teste;
  - As classes a serem testadas devem ser carregadas a partir de um arquivo jar carregado em runtime.
    - Obs.: utilize JFileChooser para seleção do arquivo jar no início da execução.

Reflexão

- Exercício 7:
  - Desenvolva um framework que converte objetos de qualquer classe para uma representação JSON
  - Cada atributo que fará parte do JSON deverá ter um método GET com uma anotação especial "@JSON".