Generics
Lab 10

O **J2SE 5.0** introduziu várias extensões à linguagem Java. Uma foi de **Generics**, que lhe permite abstração de tipos.

O exemplo mais comum são as estruturas de dados dinâmicas, como os da hierarquia de classes em **Collection**. Antes de **J2SE 5.0**, quando você extraia um elemento de uma coleção, você tinha que fazer o **cast** do tipo do elemento armazenado na coleção.

Além de ser inconveniente também é inseguro. O compilador não checa se o tipo em **cast** é o mesmo do elemento na coleção, então o **cast** pode falhar em tempo de execução. **Generics** provê uma maneira para você comunicar o tipo da coleção ao compilador, assim ele poderá fazer a checagem em tempo de compilação. Uma vez que o compilador sabe o tipo do elemento da coleção, o compilador pode checar se você está usando a coleção de forma consistente e pode associar o **cast** correto aos itens extraídos da coleção.

Neste laboratório, você vai aprender como usar **Generics** fazendo uma extensão ao modelo de classes utilizado até o momento.

Duração prevista: 60 minutos

Exercícios

Exercício 1: Construindo uma classe Genérica

Exercício 2: Generics e subtipagem

Construindo uma classe Genérica

1 - A respeito da real necessidade, vamos criar uma nova classe chamada **Movimento** que guardará todos as transações realizadas em uma **Conta** conforme o exemplo abaixo.

```
import java.util.*;

/**
 * Classe que implementa um movimento de transacoes.
 * Um movimento é apenas uma serie de transacoes feitas.
 * Todas as transacoes devem entrar aqui em ordem cronologica.
 */
public class Movimento<T> {

	// uma colecao deve manter a ordem de inserçao
	private ArrayList<T> transacoes;

	// Construtores
	/**
	 * Controi um Movimento vazio (sem transacoes).
	 */
	public Movimento() {

		this.transacoes = new ArrayList<T>();
	}

	/**
	 * Adiciona uma transacoes ao movimento.
	 */
	public void add(T transacao) {

		transacoes.add(transacao);
	}
```

```
	/**
	 * Fornece um Iterator para varrer as transacoes por data.
	 */
	public Iterator<T> getTransacoes() {

		return transacoes.iterator();
	}

}
```

2 - Crie a classe **TesteMovimento** conforme exemplo abaixo e veja o uso da nova classe. Ela funciona como um contêiner de um tipo de dados qualquer.

```
import java.util.Iterator;

public class TesteMovimento {

	public static void main(String[] args) {

		Movimento<Transacao> m1 = new Movimento<Transacao>();
		Movimento<String> m2 = new Movimento<String>();

		m1.add(new Transacao(UtilData.data(), new Conta("Fulano", 1000), new
Conta("Ciclano", 2000), 0.0, "nda", EnumTipoTransacao.TRANSFERENCIA));

		// erro compilacao
		// m1.add(new String("qq coisa"));
		m2.add(new String("nda de +"));

		Iterator it;
		it = m1.getTransacoes();
		while (it.hasNext())
			System.out.println(it.next());

		it = m2.getTransacoes();
		while (it.hasNext())
			System.out.println(it.next());
	}

}
```

Generics e subtipagem

1 - A nova classe **Movimento.java** tem muito mais um fim didático que funcional no contexto da aplicação que você está desenvolvendo ao longo destes laboratórios. Você deve modificar a classe **Conta.java** como na listagem abaixo, ou seja, o atributo da classe declarado como um **ArrayList** será substituído pela classe **Movimentacao.java** que acabou de ser criada para armazenar os movimentos da **Conta.java**.

Modifique esse trecho:

```
private ArrayList movimento;
```

Para este trecho:

```
private Movimento<Transacao> movimento;
```

Modifique esse trecho no construtor:

```
movimento = new ArrayList();
```

Para este trecho:

```
movimento = new Movimento<Transacao>();
```

A declaração da **variável movimento** agora restringe o tipo de objeto que pode ser colocado dentro do nosso contêiner **Movimento**, o que te interessa é guardar referências de objetos apenas do tipo **Transacao**. Assim se você tentar colocar qualquer outro tipo de referência no contêiner **Movimento** o compilador não irá permitir evitando erro durante a execução.

2 - Veja que a compilação informa que os métodos **getMovimento()** e **setMovimento()**, não garantem que o objeto na coleção é checado quando ao tipo, assim devemos alterar estes métodos também.

```
public ArrayList getMovimento() {

        return movimento;
}

public void setMovimento(ArrayList movimento) {

        this.movimento = movimento;
                                                                                }
```

Para:

```
public Movimento<Transacao> getMovimento() {

        return movimento;
}

public void setMovimento(Movimento<Transacao> movimento) {

        this.movimento = movimento;
}
```

3 - Uma vez modificada a classe **Conta** também devemos modificar nossas classe **ExtratoTXT e ExtratoHTML**, nesta classe nos percorremos toda a coleção de transações e imprimimos o movimento da conta. Modifique a chamada ao método **iterator()** conforme o exemplo abaixo resolvendo o problema de compilação da classe. Perceba que não é mais necessário fazer o **castin** para classe **Transacao**.

```
Iterator<Transacao> it = conta.getMovimento().getTransacoes();
while (it.hasNext()) {
        transacao t = it.next();
```

4 - Para testar a nova classe **Movimento, Conta** e **ExtratoTXT** execute a classe **ExtratoContaCorrente.**