

Lab 10 – Generics

O **J2SE 5.0**, introduziu várias extensões à linguagem Java. Uma foi de **Generics (genericidade)**, lhe permite abstração de tipos. O exemplo mais comum são os tipos container, tal como os da hierarquia de classes em **Collection**. Antes de **J2SE 5.0**, quando você extrai um elemento de uma coleção, você precisa 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. Sugerimos que estes exemplos sejam feitos com uso da IDE Eclipse, com ela os erros de programação vão sendo mostrados imediatamente à medida que você escreve o código.

Duração prevista: 70 minutos

Exercícios

Exercício 1: Construindo uma classe Genérica (30)

Exercício 2: Generics e subtipagem (40)

Exercício 1 – Construindo uma classe Genérica

1. A respeito da real necessidade, vamos criar uma nova classe chamada **Movimento.java** que guardará todas as transações realizadas em uma **Conta.java** conforme abaixo na **Listagem-9.1**.

```
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 insercao
    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();

}
```

Listagem 9.1 – classe Movimento

2. Crie a classe **TesteMovimento.java** conforme a **Listagem-9.2** e veja o uso da nova classe. Ela funciona como um container 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());

    }

}
```

Listagem 9.2 - TesteMovimento

3. 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 excluí-la após o termino desse laboratório.