**Generics Type**

* Um tipo genérico é uma classe genérica ou uma interface que é parametrizada em relação a tipos.
* A classe Box a seguir será modificada para demonstrar o conceito:

public class Box {

private Object object;

public void set(Object object) { this.object = object; }

public Object get() { return object; }

}

* O símbolo <> é chamado de "diamond" ou "diamond operator" foi um recurso introduzido no Java 7 e é usado no contexto de tipos genéricos em Java para inferir automaticamente o tipo com base no contexto.
* Para atualizar a classe Box para usar generics, você cria uma declaração de tipo genérico alterando o código public class Box para public class Box<T>.
* Isso introduz a variável de tipo, T, que pode ser usada em qualquer lugar dentro da classe:

/\*\*

Versão genérica da classe Box.

@param <T> o tipo do valor sendo armazenado

\*/

public class Box<T> {

// T representa "Type" (tipo)

private T t;

public void set(T t) { this.t = t; }

public T get() { return t; }

}

* Como você pode ver, todas as ocorrências de Object são substituídas por T.
* Uma variável de tipo pode ser qualquer tipo não primitivo que você especificar: qualquer tipo de classe, qualquer tipo de interface, qualquer tipo de array ou até mesmo outra variável de tipo.
* Essa mesma técnica pode ser aplicada para criar interfaces genérica.
* **Os nomes de parâmetros de tipo mais comumente usados são:**
  + E - Elemento (usado extensivamente pelo Java Collections Framework)
  + K - Chave
  + N - Número
  + T - Tipo
  + V - Valor
  + S, U, V, etc. - 2º, 3º, 4º tipos

**Vantagens simples de usar generics nas interfaces Collection em Java:**

1. Segurança do tipo de dados: O uso de generics garante que apenas objetos de um tipo específico possam ser adicionados à coleção, evitando erros de tipo e garantindo que você esteja lidando com os dados corretos.
2. Código mais legível: Ao usar generics, você pode especificar o tipo de dados esperado ou retornado pela coleção, o que torna o código mais fácil de entender e ler.
3. Detecta erros mais cedo: O compilador verifica se você está usando os tipos corretos durante a compilação, ajudando a identificar erros de tipo antes mesmo de executar o programa.
4. Reutilização de código: Com generics, você pode criar classes e métodos genéricos que funcionam com diferentes tipos de coleções, evitando a necessidade de duplicar código para cada tipo específico.
5. Melhor desempenho: O uso de generics pode melhorar o desempenho, pois evita a necessidade de conversões de tipo desnecessárias e permite que o compilador otimize o código com base no tipo especificado.