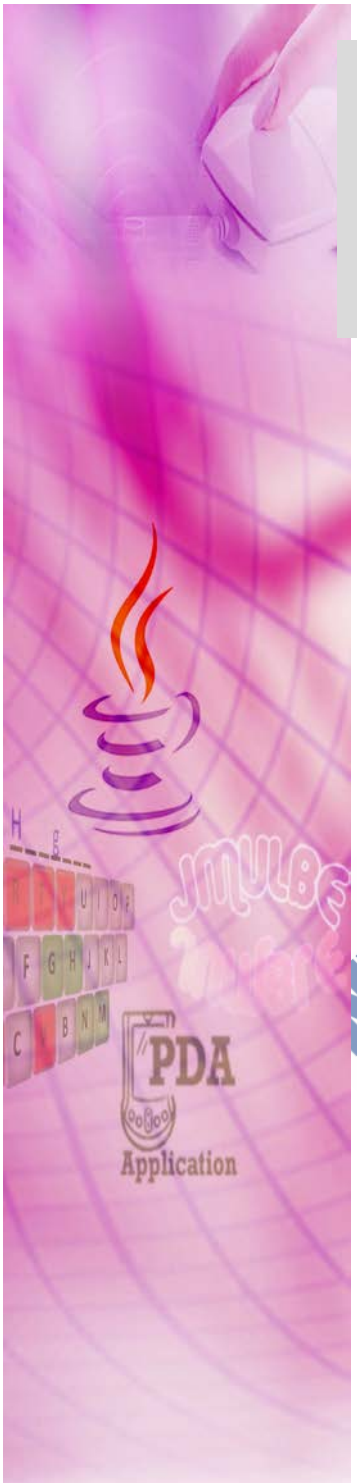


第 3 章

使用泛型

有时，您需要创建可使用不同类型数据的接口、类和方法。为此，你需要编写强大的类型安全性代码，这是一项冗长且容易出错的任务。要克服编写类型安全性代码的开销，Java 提供了 *generics* 功能。泛型允许您创建泛化的接口、类和方法。

本章重点介绍用户定义的泛型类和方法的创建以及类型安全性的实现。

目标

在本章中，您将学习：

- 创建用户定义的泛型类和方法
- 实现类型安全性


NIIT
training
china
.com

创建用户定义的泛型类和方法

思考场景：您需要使用一种返回多种对象（例如 Integer、String 和 Double）的方法开发类。要实现该功能，您需要创建泛型类和方法。泛型表示使您能够编写代码以用于多种类型对象的参数化类型。此外，还允许您对类和方法进行泛化以增强它们的功能。创建泛型类和方法是为了使引用和方法能够接受任何类型的对象。

创建泛型类

泛型使您能够泛化类。这意味着引用变量、方法的自变量和返回类型可以是任何类型。在泛型类声明中，类名后跟*类型参数*部分。类型参数部分由尖括号 < > 表示，其中可以包含一种或多种类型的参数，以逗号分隔。以下语法用于创建泛型类：

```
class [ClassName]<T>
{

}
```

您也可以使用以下代码段来创建泛型类：

```
class GenericClassDemo<T>
{

}
```

思考一个示例：您需要创建泛型类来设置和获取 Integer 和 String 值。为此，您可以使用以下代码：

```
public class GenericClassDemo<T>
{
private T t;

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

public T getValue()
{
return t;
}

public static void main(String[] args)
{
GenericClassDemo<Integer> iobj = new GenericClassDemo<Integer>();
iobj.setValue(10);
System.out.println(iobj.getValue());

GenericClassDemo<String> sobj = new GenericClassDemo<String>();
sobj.setValue("Ten");
```

```
System.out.println(sobj.getValue());
}
}
```

执行上述代码时，会显示以下输出：

```
10
Ten
```

在上述代码中，GenericClassDemo 类是通过两种方法 setValue() 和 getValue() 创建的。T 表示数据类型，可以接受任何对象，例如 Integer 和 String。上述代码中的 T 参数用作引用类型、返回类型和方法自变量。

创建泛型方法

在泛型类中，方法可以使用类的类型参数，这样会使方法自动成为泛型。思考泛型类内部的泛型方法的以下代码：

```
public class GenericClassDemo<T>
{
private T t;

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

public T getValue()
{
return t;
}
}
```

在上述代码中，setValue() 和 getValue() 方法使用类的类型参数。

但是，您可以声明包含其自己的一个或多个类型参数的泛型方法。泛型方法的声明 包含由尖括号 < > 表示的类型参数。以下语法用于创建泛型方法：

```
public <Type Parameter> [Return Type] [MethodName](Argument list…)
{
}
```

您可以创建泛型方法，如以下代码段所示：

```
public <T> T showValue(T val)
{
}
```

类型参数部分可以具有一种或多种类型的参数，以逗号分隔。此外，还可以在非泛型类中创建泛型方法。

思考非泛型类内部的泛型方法的以下代码：

```
public class GenericMethodDemo
{
public <M> M display(M val)
{
return val;
}

public static void main(String[] args)
{
GenericMethodDemo obj = new GenericMethodDemo();

System.out.println("The generic method is called with String value:" +
obj.display("Test"));
System.out.println("The generic method is called with Double value:" +
obj.display(7.5));
System.out.println("The generic method is called with Boolean value:" +
obj.display(true));
System.out.println("The generic method is called with Integer value:" +
obj.display(10));

}
}
```

执行上述代码时，会显示以下输出：

```
The generic method is called with String value:Test
The generic method is called with Double value:7.5
The generic method is called with Boolean value: true
The generic method is called with Integer value:10
```

在上述代码中，`GenericMethodDemo` 类包含用于显示值的 `display()` 方法。此外，在 `main()` 方法中，对四种不同类型的值调用四次显示方法。

实现类型安全性

思考您要创建泛型类对象的示例。要创建该对象，需要调用具有所需类型参数（例如 `Integer` 或 `String`）的泛型类构造函数以增加代码长度。但是，要缩短编码长度，Java 提供了一定的灵活性，即只要编译器能够根据上下文猜出或判断自变量类型就允许保留类型参数为空。此外，Java 还提供了允许在类型参数中实现继承的通配符。

使用泛型类型推断

您已学习如何创建泛型类和方法。要使用这些类或方法，需要了解 Java 如何推断提供为类型参数的自变量。在使用泛型类型时，必须为针对泛型类型声明的每个类型参数提供类型自变量。类型自变量列表是跟在类型名后通过尖括号定界的逗号分隔列表。例如，思考具有两个类型参数的泛型类型的以下代码：

```
class Pair<X, Y>
{

private X first;
private Y second;

public Pair(X a1, Y a2)
{
first = a1;
second = a2;
}

public X getFirst()
{
return first;
}

public Y getSecond() {
return second;
}

public static void main(String[] args)
{
Pair<String, Integer> obj1 = new Pair<String, Integer>("Test", 1);
System.out.println("String is " + obj1.getFirst());
System.out.println("Integer is " + obj1.getSecond());

Pair<Integer, String> obj2 = new Pair<Integer, String>(2, "Demo");
System.out.println("Integer is " + obj2.getFirst());
System.out.println("String is " + obj2.getSecond());

}
}
```

执行上述代码时，会显示以下输出：

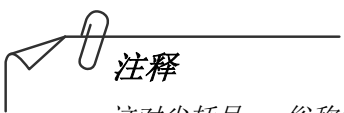
```
String is Test
Integer is 1
Integer is 2
String is Demo
```

在上述代码中，`Pair` 类包含两个方法 `getFirst()` 和 `getSecond()`，分别返回 `first` 和 `second` 变量的值。在 `main()` 方法中，创建了两个 `<String, Integer>` 和 `<Integer, String>` 类型的对象。

Java 提供了新功能*类型推断*，使您能够使用空类型参数集 `<>` 调用泛型类型的构造函数。只要编译器能够根据上下文推断出类型自变量就可以使用这个空的类型参数集。

思考上述示例，您可以在创建 `Pair` 类的对象时使用空类型参数集，如以下代码段所示：

```
Pair<String, Integer> obj1 = new Pair<>("Test", 1);

Pair<Integer, String> obj2 = new Pair<>(2, "Demo");
```



注释

这对尖括号 `<>` 俗称钻石运算符。

使用通配符

泛型遵循简单的声明规则，即引用变量声明类型必须与传递到实际对象的类型匹配。这意味着在泛型中，子类不能作为超类的子类型传递，如以下代码段所示：

```
WildCardDemo<Number> obj = new WildCardDemo<Integer>();
```

泛型仅允许以下类型的声明：

```
WildCardDemo<Integer> obj = new WildCardDemo<Integer>();
```

或

```
WildCardDemo<Number> obj = new WildCardDemo<Number>();
```

但是，Java 提供了通配符自变量以在泛型中将子类作为超类的子类型传递。通配符自变量是泛型类型实例中用作类型自变量的特殊构造。有多种类型的通配符：

下表列出了最常用的通配符自变量：

通配符	*描述*
? extends	*它指定类型可以是扩展自 extends 关键字后面所指定的类的任何类型。它是有限制的通配符。*
? super	*它指定类型可以是作为 super 关键字后面所指定类的基类的任何类型。它是有限制的通配符。*
?	*它指定类型可以是任何类型。它不需要是任何类的子类型或超类型。它是没有限制的通配符。*

通配符自变量

注释

有限制通配符用于通过关键字限制任何类型的对象：extends、implements 和 super。无限制通配符不绑定任何类型的对象，并且不与任何关键字关联。

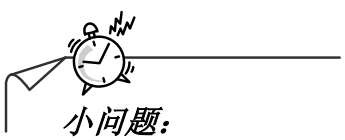
思考您要比较两个对象的值的示例。但是，您只需要比较数字对象。为此，您可以使用以下代码：

```
public class WildCardDemo<T> {

private T t;

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

public T getValue() {
return t;
}
public boolean compare(WildCardDemo<? extends Number> wcd)
{
if(t == wcd.t)
{
return true;

}
else
{
return false;
}

}
```

```
public static void main(String[] args)
{
WildCardDemo<Integer> obj1 = new WildCardDemo<Integer>();
obj1.setValue(10);

WildCardDemo<String> obj2 = new WildCardDemo<String>();
obj2.setValue("Test");

System.out.println("Value of first object:" +obj1.getValue());
System.out.println("Value of second object:" +obj2.getValue());


System.out.println("Are both equal?" + obj1.compare(obj2));  //Compilation
Error

}
}
```

在上述代码中，使用三种方法创建 `WildCardDemo` 类：`setValue()`、`getValue()` 和 `compare()`。`compare()` 方法使用通配符以使 `Number` 的对象或其一个子类型可在运行时传递到该对象。调用 `compare()` 方法时，发生编译错误，因为 `obj2` 的类型为 `String`，而 `obj1` 的类型为 `Integer`。

在上述代码中，使用 `<? extends Number>` 通配符，告知指定对象必须继承 `Number` 类。



小问题：

以下哪个通配符用于指定从指定类继承的类型？

1. *`<?>`*
2. *`<? extends >`*
3. *`<extends ?>`*
4. *`<? extends ?>`*

答案：

2.`<? extends >`

活动 3.1：使用泛型

练习问题

1. 以下哪个符号用于定义类型参数？
   a. `{ }`
   b. `[ ]`
   c. `( )`
   d. `< >`

2. 说明以下语句是正确还是错误。
   非泛型类可以包含泛型方法。

3. 指出泛型类的正确语法。
   a. `public <T> class Myclass { }`
   b. `public class <T> Myclass { }`
   c. `public class Myclass <T> { }`
   d. `public class Myclass {<T>}`

4. 指出泛型方法的正确语法。
   a. `public <T> T myMethod()`
   b. `public myMethod<T> T()`
   c. `public <T> myMethod()`
   d. `public myMethod T()`

5. 以下哪个通配符指定类型可以为任意类型？
   a. `?`
   b. `? extends`
   c. `? super`
   d. `? implements`

小结

在本章中，您学习了：

- 泛型使您能够泛化类，这意味着引用变量、方法的自变量和返回类型可以是任何类型。
- 类型参数部分由尖括号 < > 表示，其中可以包含一种或多种类型的参数，以逗号分隔。
- 在泛型类中，方法可以使用类的类型参数，这样会使方法自动成为泛型。
- 您可以声明包含其自己的一个或多个类型参数的泛型方法。
- 泛型方法的声明 包含由尖括号 < > 表示的类型参数。
- Java 提供了允许在类型参数中实现继承的通配符。
- Java 提供了新功能类型推断，使您能够使用空类型参数集 <> 调用泛型类型的构造函数。
- Java 提供了通配符自变量以在泛型中将子类作为超类的子类型传递。
- Java 支持以下类型的通配符：
  - `? extends`
  - `? super`
  - `?`