Advanced Programming

Generic

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

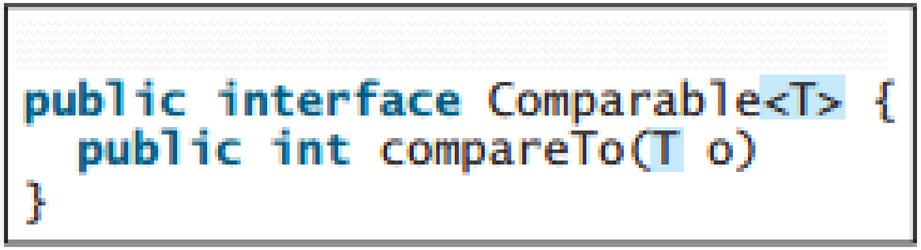
Introduction

- *Generics* is the capability to parameterize types.
  - Define a class or a method with generic types that the compiler can replace with concrete types.
- For example:
  - Define a generic stack class that stores the elements of a generic type.
  - From this generic class, you may create a stack object for holding **strings** and a stack object for holding **numbers**.
    - Strings and numbers are concrete types that replace the generic type.

Introduction

- The key benefit of generics is to enable errors to be detected at compile time rather than at runtime.
  - A generic class or method permits you to specify allowable types of objects that the class or method may work with.
  - If you attempt to use an incompatible object, the compiler can detect the errors.

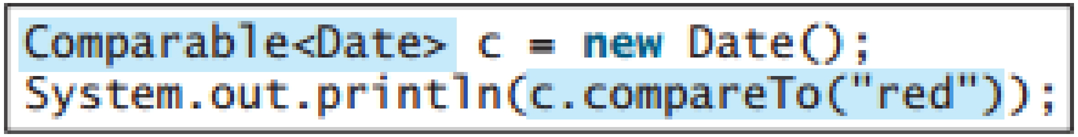
Motivations and Benefits



```
public interface Comparable<T> {
  public int compareTo(T o)
}
```

- <T> represents a *formal generic type*, which can be replaced later with an *actual concrete type*.
- Replacing a generic type is called a *generic instantiation*.
- By convention, a single capital letter such as **E** or **T** is used to denote a formal generic type.

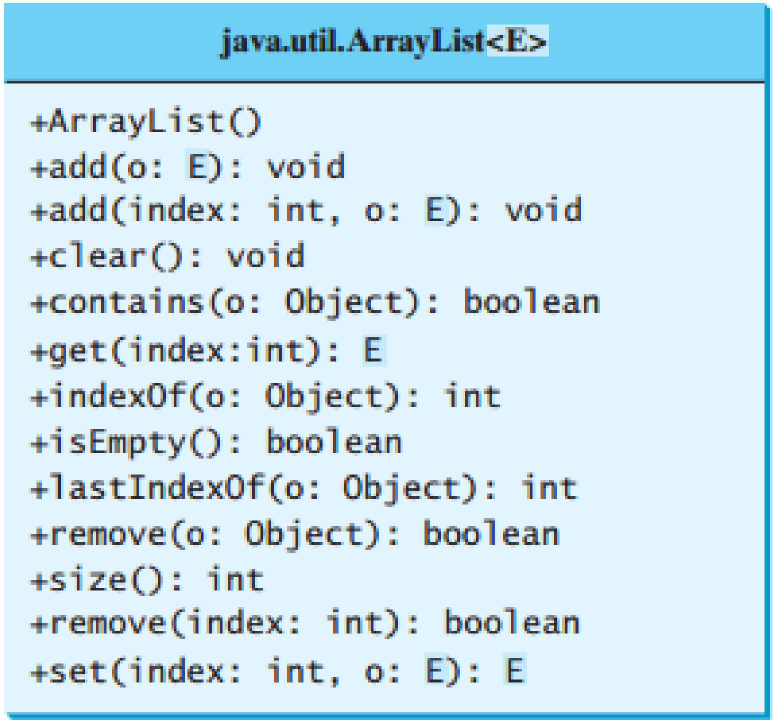
Example



```
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

- **c** is a reference variable whose type is **Comparable** and invokes the **compareTo** method to compare a **Date** object with a string.
- The code has a compile error, because the argument passed to the **compareTo** method must be of the **Date** type.
  - The errors can be detected at compile time rather than at runtime, the generic type makes the program more reliable.

ArrayList



java.util.ArrayList<E>

```
+ArrayList()
+add(o: E): void
+add(index: int, o: E): void
+clear(): void
+contains(o: Object): boolean
+get(index:int): E
+indexOf(o: Object): int
+isEmpty(): boolean
+lastIndexOf(o: Object): int
+remove(o: Object): boolean
+size(): int
+remove(index: int): boolean
+set(index: int, o: E): E
```

ArrayList Example

- Example: **ArrayList** list = **new** **ArrayList** ();
  - You can add *only strings* into the list.

    list.add("**Red**");
- Generic types must be reference types, cannot replace a generic type with a primitive type (**int**, **double**, or **char**)

  ArrayList<~~int~~> intList = **new** ArrayList<~~int~~> ();
- To create an **ArrayList** object for **int** values:

  ArrayList<Integer> intList = **new** ArrayList<Integer> ();
  - You can add an **int** value to list: intList.add(5);
  - Java wraps **5** into new **Integer(5)**

ArrayList Example

- If the elements are of wrapper types, such as **Integer**, **Double**, and **Character**, you can directly assign an element to a primitive type variable.

```
ArrayList<Double> list = new ArrayList<Double>();
list.add(5.5); // 5.5 is automatically converted to new Double(5.5)
list.add(3.0); // 3.0 is automatically converted to new Double(3.0)
Double doubleObject = list.get(0); // No casting is needed
double d = list.get(1);// Automatically converted to double
```

Defining Generic Classes and Interfaces

```
public class GenericStack<E> {
        ArrayList<E> list = new ArrayList<E>();

        public int getSize() {
                return list.size();
        }
        public E peek() {
                return list.get(getSize() - 1);
        }
        public void push(E o) {
                list.add(o);
        }
```

Defining Generic Classes and Interfaces

```
        public E pop() {
                E o = list.get(getSize() - 1);
                list.remove(getSize() - 1);
                return o;
        }
        public boolean isEmpty() {
                return list.isEmpty();
        }
}
```

Defining Generic Classes and Interfaces

```
public static void main(String[] args) {
        GenericStack stack1 = new GenericStack ();
        stack1.push("London");
        stack1.push("Paris");
        stack1.push("Berlin");

        GenericStack stack2 = new GenericStack ();
        stack2.push(1); // auto boxing 1 to new Integer(1)
        stack2.push(2);
        stack2.push(3);
}
```

Generic Methods

```
public class GenericMethodDemo {
   public static void main(String[] args) {
      Integer[] integers = { 1, 2, 3, 4, 5 };
      String[] strings = { "London", "Paris", "New York",
"Austin" };

      GenericMethodDemo.<Integer> print(integers);
      GenericMethodDemo.<String> print(strings);
   }
   public static <E> void print(E[] list) {
      for (int i = 0; i < list.length; i++)
             System.out.print(list[i] + " ");
      System.out.println();
   }
}
```

Bounded Type

- A generic type can be specified as a subtype of another type. Such a generic type is called *bounded*

Bounded Type

```
public class BoundedTypeDemo {
   public static void main(String[] args) {
      Rectangle rectangle = new Rectangle(2, 2);
      Circle circle = new Circle(2);
      System.out.println("Same area? " +
      BoundedTypeDemo.<GeometricObject>equalArea(rectangle,
circle));
   }

   public static <E extends GeometricObject> boolean
equalArea (E object1, E object2) {
      return object1.getArea() == object2.getArea();
   }
}
```

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.