BIRZEIT UNIVERSITY

Generics

Liang, Introduction to Java Programming and Data Structures,
Twelfth Edition, (c) 2020 Pearson Education, Inc. All rights reserved.

By: Mamoun Nawahdah (Ph.D.)

2022/2023

Objectives

- To describe the benefits of generics
- To use generic classes and interfaces
- To define generic classes and interfaces
- To explain why generic types can improve reliability and readability
- To define and use generic methods and bounded generic types
- To develop a generic sort method to sort an array of Comparable objects

STUDENTS-HUB.com

Motivations and Benefits

- 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 can work with.
- If you attempt to use an incompatible object, the compiler will detect that error.

Comparable interface

Here, **<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.

```
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5

```
package java.lang;

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

(b) JDK 1.5

STUDENTS-HUB.com

Reliable code

```
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5

Runtime error

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

(a) Prior to JDK 1.5

```
package java.lang;

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

(b) JDK 1.5

Compile error

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

(b) JDK 1.5

Since the errors can be detected at compile time rather than at runtime, the generic type makes the program more **reliable**.

ArrayList

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

(a) ArrayList before JDK 1.5

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

(b) ArrayList since JDK 1.5

STUDENTS-HUB.com

Generic reference type

❖ Generic types must be reference types. You cannot replace a generic type with a primitive type such as **int**, **double**, or **char**.

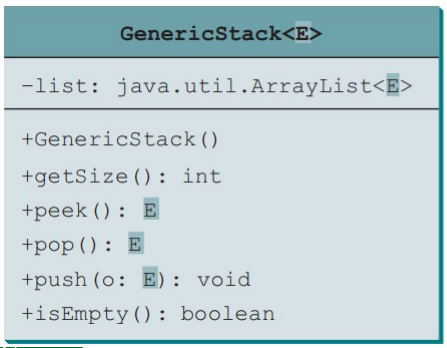
ArrayList<**int**> intList = new ArrayList<>(); X

```
ArrayList<Double> list = new ArrayList<>();
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

❖ A generic type can be defined for a class or interface.

❖ A concrete type must be specified when using the class to create an object or using the class or interface to declare a reference variable.



STUDENTS-HUB.com

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

  public int getSize() {
    return list.size();
  }
  public E peek() {
    return list.get(getSize() - 1);
  }
  public void push(E o) {
    list.add(o);
  }
  public E pop() {
    E o = list.get(getSize() - 1);
    list.remove(getSize() - 1);
    return o;
  }
  public boolean isEmpty() {
    return list.isEmpty();
  }
  public String toString() {
    return "stack: " + list.toString();
  }
}
```

Benefits of using generic types

```
GenericStack<String> stack1 = new GenericStack<>();
stack1.push("London");
stack1.push("Paris");
stack1.push("Berlin");

GenericStack<Integer> stack2 = new GenericStack<>();
stack2.push(1); // autoboxing 1 to an Integer object
stack2.push(2);
stack2.push(3);
```

- Instead of using a generic type, you could simply make the type element **Object**, which can accommodate any object type.
- However, using a specific concrete type can improve software **reliability** and **readability** because certain errors can be detected at compile time rather than at runtime.

STUDENTS-HUB.com

Multiple generic parameters

- Occasionally, a generic class may have more than one parameter.
- In this case, place the parameters together inside the brackets, separated by commas

<E1, E2, E3>

Inheritance with generics

- You can define a class or an interface as a subtype of a generic class or interface.
- For example, the **String** class is defined to implement the Comparable interface in the Java API as follows:

public class String implements Comparable<String>

STUDENTS-HUB.com

Generic Methods

- A generic type can be defined for a **static** method.

```
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 generic type

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

```
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? " +
      equalArea(rectangle, circle));
  }

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

STUDENTS-HUB.com

Case Study: Sorting an Array of Objects

```java
/** Sort an array of comparable objects */
public static <E extends Comparable<E>> void sort(E[] list) {
  E currentMin;
  int currentMinIndex;

  for (int i = 0; i < list.length - 1; i++) {
    // Find the minimum in the list[i+1..list.length-2]
    currentMin = list[i];
    currentMinIndex = i;

    for (int j = i + 1; j < list.length; j++) {
      if (currentMin.compareTo(list[j]) > 0) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Case Study: Sorting an Array of Objects

```java
public static void main(String[] args) {
  // Create an Integer array
  Integer[] intArray = {Integer.valueOf(2), Integer.valueOf(4),
    Integer.valueOf(3)};

  // Create a Double array
  Double[] doubleArray = {Double.valueOf(3.4), Double.valueOf(1.3),
    Double.valueOf(-22.1)};

  // Create a Character array
  Character[] charArray = {Character.valueOf('a'),
    Character.valueOf('J'), Character.valueOf('r')};

  // Create a String array
  String[] stringArray = {"Tom", "Susan", "Kim"};

  // Sort the arrays
  sort(intArray);
  sort(doubleArray);
  sort(charArray);
  sort(stringArray);
```

STUDENTS-HUB.com

Wildcard Generic Types

- You can use unbounded wildcards, bounded wildcards, or lower bound wildcards to specify a **range** for a generic type.
- A wildcard generic type has three forms:
  - **Unbounded wildcard**, <**?>**, is the same as **? extends Object**.
  - **Bounded wildcard**, <**? extends T>**, represents **T** or a **subtype** of **T**.
  - **Lower bound wildcard**, <**? super T>**, denotes **T** or a **supertype** of **T**.

STUDENTS-HUB.com