Chapter 19 Generics

Why Do You Get a Runtime Error?

```
package java.lang;
public interface Comparable {
    public int compareTo(Object o)
}
```

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

Fix the Error

```java
package java.lang;
public interface Comparable<T> {
    public int compareTo(T o)
}
```

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

What is Generics?

- *Generics* is the capability to parameterize types.
- With this capability, you can define a class or a method with generic types that can be substituted using concrete types by the compiler.
- For example, you may 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. Here, strings and numbers are concrete types that replace the generic type.

Why Generics?

- 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 the class or method with an incompatible object, the compile error occurs.

Generic ArrayList before and after JDK 1.5

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 in JDK 1.5

Using Generic ArrayList

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

The customized Arraylist code replacing the generic type ***E*** with ***String*** in this case is as follows:

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

```
list.add("Red");
```

```
list.add(new Integer(1));
```

No Casting Needed

```
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
```

Practice Exercises

19.1 Are there any compile errors in (a) and (b)?

```
ArrayList dates = new ArrayList();
dates.add(new Date());
dates.add(new String());
```

(a) Prior to JDK 1.5

```
ArrayList<Date> dates =
  new ArrayList<>();
dates.add(new Date());
dates.add(new String());
```

(b) Since JDK 1.5

19.2 What is wrong in (a)? Is the code in (b) correct?

```
ArrayList dates = new ArrayList();
dates.add(new Date());
Date date = dates.get(0);
```

(a) Prior to JDK 1.5

```
ArrayList<Date> dates =
  new ArrayList<>();
dates.add(new Date());
Date date = dates.get(0);
```

(b) Since JDK 1.5

Declaring Generic Classes and Interfaces

GenericStack<E>	
-list: java.util.ArrayList<E>	An array list to store elements.
+GenericStack()	Creates an empty stack.
+getSize(): int	Returns the number of elements in this stack.
+peek(): E	Returns the top element in this stack.
+pop(): E	Returns and removes the top element in this stack.
+push(o: E): E	Adds a new element to the top of this stack.
+isEmpty(): boolean	Returns true if the stack is empty.

Generic Methods

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

```
public static void print(Object[] list) {
  for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");
  System.out.println();
 }
```

Bounded Generic Type

```
public static void main(String[] args ) {
    Rectangle rectangle = new Rectangle(2, 2);
    Circle9 circle = new Circle9(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();
}
```