Lecture 4: Generics

Reading materials
Dale, Joyce, Weems: generics are assumed in the book so all chapters that we covered are using generics, but basics are not discussed
OpenDSA: not discussed
Liang: only in Comprehensive edition
The course website has links to some additional reading materials about generics.

Topics Covered

1 What are Java Generics

2 Generics in Java Libraries
   2.1 ArrayList <E> class
   2.2 Comparable <E> interface
   2.3 Other generic containers

3 Defining Generic Methods

4 Restricting Generics

5 Defining Generic Classes and Interfaces
1 What are Java Generics

Generics (known as templates in other programming languages) have been introduced to Java in version 1.5. They provide capability of parametrizing types when defining methods and classes.

Example

- We looked at methods that sort integers, doubles, Strings and other objects.
- But for each type we need a different method, even though they are essentially the same.
- Generics allow us to write one sort method that can sort pretty much anything.
- Just like Arrays.sort() does.

In generic definitions, we use a formal generic type name, for example <T> that ”later” is replaced by the compiler by an actual type.

- Convention is to use capital letters T or E to stand for generic type, but you are free to use other upper case letters..
- You can use multiple generic type placeholders in a single definition. Simply put all of them inside the angled brackets like this <T, E, S>.

The formal generic type stands for any (suitable) reference type. You cannot use primitive types in place of generic type placeholders. If you are using generics, you need to use reference types!

2 Generics in Java Libraries

2.1 ArrayList <E> class

[See chapter 11 in Liang.]

We have been using an ArrayList class from Java libraries without really calling it a generic type. ArrayList objects can store any types of data. The full name of the type is really ArrayList<E> and when you define and create these list objects you need commit what type of data you want to store in it.

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

creates a list of String objects, and

```
ArrayList <Integer> numberList = new ArrayList<Integer>();
```

creates a list of Integer objects.

Once the list is created (declared and instantiated) you do not really need to worry that it is a generic list.

Note: You cannot create an ArrayList object for which the type of data is a primitive type (int, double, char, etc).
2.2 Comparable <E> interface

[See chapter 15 in Liang.]

We also have been using a comparable interface. The full name of that interface is Comparable<E>. If you are defining a class that implements Comparable, then you have to specify "what type" of Comparable object you want. That will usually match the name of the class that is being defined. This makes sure that you are not trying to compare proverbial apples with oranges.

Example:

```java
public class Circle implements Comparable<Circle> {
    //Circle class definition
    public int compareTo(Circle c) {
        if (this.r > c.r)
            return 1;
        else if (this.r < c.r)
            return -1;
        else
            return 0;
    }
}
```

2.3 Other generic containers

There are many other generic classes in the libraries that you might have encountered:

- Stack<E> class, [http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html](http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html)
- LinkedList<E> class, [http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html)
- Queue<E> interface, [http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html](http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html)
- ...

3 Defining Generic Methods

You can create generic methods in classes that are otherwise not generic themselves. It is not significantly different than defining ordinary methods, except for one "funny" syntax that you have to remember about.

```java
accessModifiers <T> returnType methodName ( parameterList ) {
    ...
}
```
You need to indicate that the method is going to be generic by writing the generic placeholder in angled brackets between access modifiers and the return type. The parameter list, body of the method and the return type can contain generic placeholder(s) in place of reference types.

For example, the following method can be used to print the content of any type of an array:

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

(as long as the `toString()` method of the given types does something meaningful).

### 4 Restricting Generics

Sometimes you may want to restrict the types that are used with your generic method or class. For example, it does not make sense to sort things that do not provide implementation of `compareTo()` method. You can specify what interface the actual type has to implement to be acceptable. To do that, we use another “funny” syntax:

- `< E extends Clonable >` - specifies that actual types used in place of `E` have to be implementing the `Clonable` interface provided by Java.

- `< E extends Comparable <E> >` - specifies that actual types used in place of `E` have to be implementing the `Comparable<E>` interface (this means they implement generic comparable type).

See `ArrayTools.java` class and `GenericArrayTools.java` for examples of defining and using generic methods.

### 5 Defining Generic Classes and Interfaces

Defining a generic class is not that different than defining any other class. We need to decide on which is the generic type and use the place holder for it instead of the actual name of the type.

We will start using generic classes in a few weeks.