Lecture 6: 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

2 Generics in Java Libraries 2
   2.1 ArrayList <E> class .............................................. 2
   2.2 Comparable <E> interface ...................................... 3
   2.3 Other generic containers ....................................... 3

3 Definining Generic Classes and Interfaces 3
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 parameterizing types.

Example

- we have been creating lists of strings, lists of integers, lists of floats, lists of spaces (for the maze implementation)
- what if we had a list type that can store any of the above (well, not all mixed together, but something that could store integers, or floats, or strings, or spaces)?
- the answer to this problem is a generic list

In generic definitions, we use a formal generic type name, for example <T> that later is replaced 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 class definition. Simply put both of them inside the angled brackets like this <T, E, S>.

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 complete 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). In fact, primitive types are not acceptable substitutes for the formal generic types. If you are using generics, you need to use reference types!
2.2 Comparable <E> interface

[See chapter 15 in Liang.]

We also have been using a comparable interface: either formally declaring our own classes to implement Comparable or simply providing a compareTo() method for a user defined class.

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.getRadius () )
            return 1;
        else if ( this.r < c.getRadius () )
            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
- LinkedList<E> class, 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
- ...

3 Definining 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 first rewrite the list interface as a generic interface (so we do not have to keep deciding what type of the list we want).
public interface GenericList <T> {
    void insert ( T item );
    void remove ( T item );
    void clear ( );
    boolean contains ( T item );
    int indexOf ( T item );
    T get ( int index );
    int size ( );
    String toString ( );
}

An implementation of this generic list can be either array based (you can think of ArrayList<T> as such) or a linked list based.

See GenericList.java, GenericNode.java, GenericLinkedList.java, GenericListBuilder.java in the source code examples for lecture 6 for the implementation and sample program.