Introduction to generics

You know that int and String array variables are declared as follows:

```
int[]  ia;          // We show the type of array elements in red
String[] sa;
```

But, before 2004, to declare an ArrayList variable one had to write

```
ArrayList al;
```

and all values added to that ArrayList were automatically cast to type Object. Suppose ArrayList al was to contain only Strings. A programmer could easily make a mistake and add a value of some other type to al. Even if no mistake was made, when a value was retrieved from al, it had to be cast back explicitly to String, e.g.

```
String s= (String)(al.get(0);
```

Having different types of arrays was easy; having different types of ArrayLists was harder and more error prone.

Generics in Java 5.0

In 2004, the new version of Java, Java 5.0, introduced *generics*, allowing one to write

```
ArrayList<Integer> al;      // We highlight the type of ArrayList elements
ArrayList<String> as;
```

thus putting the use of ArrayLists more on a par with arrays. The syntax is different but the goal is the same: allow the programmer to state in the declaration what kind of values a list of values must contain.

Obviously, the declaration of class ArrayList must also allow the programmer to state that the class has a "type parameter", and this can be done using the notation

```
public class ArrayList<E> { ... }
```

where the yellow-highlighted E stands for the class-type that the user will give in a declaration. E is a *type* parameter. This will all be explained in later discussions of generics.

How are generics implemented in Java?

There were several proposals for implementing a more detailed type system in Java. The one that was accepted had a special property called *type erasure*. Suppose a program is syntactically correct —it follows the rules not only of earlier Java but also the new rules for generics. Then, before compiling, all the generic annotations can be removed and the program compiled without them. This means that the JVM (Java virtual machine) did not have to be changed! Only the compiler had to be changed to check the new generic rules and then delete all mention of them. This system is quite different from that of some other languages, like C# (developed by Microsoft).

Why the word *generic*?

The online Merriam-Webster defines *generic* this way:

> **1:** an element of a compound proper name that is general …

where *general* means

> **2:** involving or belonging to every member of a class, kind, or group**:**
> applicable to every one in the unit referred to**:** not exclusive or excluding

An example of the use of *generic* outside computer programming is the generic drug. A manufacturer with a patent may manufacture a particular drug under their own brand-name. When the patent runs out, other companies can sell the same drug as a *generic* drug, with no brand-name and using only the chemical name for the drug, usually at a far less price. Insurance companies may request that you buy the generic version of the drug to lower costs.

In Java, think of type ArrayList as a generic type that is applicable to any class C for which you want to write ArrayList<C>. On Wikipedia, you can find,

> *Generic programming* is a style of computer programming in which algorithms are written in terms of types to be specified later, which are then instantiated when needed for specific types …