1 Why do we need generics?

Think of a data structure that holds several items. For instance, a Bookcase may have 3 shelves. Each shelf can hold one object.

```
public class Bookcase {
  private Object topshelf;
  private Object middleshelf;
  private Object bottomshelf;

  public void addtop(Object item) { topshelf = item; }
  public void makemiddle(Object item) { middleshelf = item; }
  public void addbottom(Object item) { bottomshelf = item; }

  public Object gettop() { return topshelf; }
  public Object getmiddle() { return middleshelf; }
  public Object getbottom() { return bottomshelf; }
}
```

This class gives the programmer the flexibility to mix different data types in the same Bookcase or re-use the same code for a Book-filled Bookcase and a Toy-filled Bookcase. The disadvantage is that if you want just one data type in the list, you won't get any help from Java to enforce this. For example, if you wanted to insert only Book objects in the Bookcase, there is nothing to prevent other objects such as Shoes from being added to the Bookcase if it is declared to hold Objects. Additionally, you have to make a cast every time you take an item out of the Bookcase.

```
Bookcase shelves = new Bookcase();
shelves.addtop(new Book("Harry Potter");
shelves.addbottom(new SmellySock()); // well, nothing stopped you...
Book book = (Book) shelves.gettop(); // and the cast is annoying
```

One possible solution to this problem is to declare the contents to be of type Book. This will force the Bookcase to hold only Book objects. The disadvantage is that you need a different implementation for each type of Bookcase that you want. You might also want to store Photo instances on a Bookcase.

This is where Java's generics can be extremely useful. Java's generics allow the programmer to define one data structure to be used for different data types in different applications.

Generics involve a bit of extra work when declaring or initializing a variable but remove all the annoying casts when using the generic variable later on. They also add some safety and allow Java to catch a few more code errors you might make.

2 How to make a class generic

This can be done in several easy steps

2.1 Make the overall class generic

Pick your favorite letter, I'll pick T. Modify the class declaration so it changes from

```
public class Bookcase { ... }
```

to

```
public class Bookcase<T> { ... }
```

This tells Java, the Bookcase will hold only Ts, whatever T may be. You can make both interfaces and classes generic this way.

```
public interface Shelf<T> { ... }
public class Bookcase<T> implements Shelf<T> { ... }
```

2.2 Make all the the contents generic

You will now replace the contents (Object) with T. To do this, find all occurences of Object and make them use T instead.

```
  public void addtop(T item) { topshelf = item; }
  public void addmiddle(T item) { middleshelf = item; }
  public void addbottom(T item) { bottomshelf = item; }

  public T gettop() { return topshelf; }
  public T getmiddle() { return middleshelf; }
  public T getbottom() { return bottomshelf; }
```

3 How to use a generic type

To make and use a non-generic Bookcase of Books, you originally used

```
Bookcase shelves = new Bookcase();
shelves.addtop(new Book("Harry Potter");
Book book = (Book) shelves.gettop();
```

Now, you declare the Bookcase to hold only Books and you can drop the cast when you retrieve a Book.

```
Bookcase<Book> shelves = new Bookcase<Book>(); // notice we used <Book> twice
shelves.addtop(new Book("Harry Potter");
Book book = shelves.gettop(); // no more casts!
```

4 How to use a generic type inside another generic type

Generic classes frequently contain uses of other generic classes. For example, Russian babushka dolls are wooden painted dolls that contain smaller and smaller dolls. If the dolls also contained an object, a class for these might look like

```
public class Babushka {
  private Object contents;
  private Babushka next;
}
```

The generic version of this class adds <T> to the class declaration and replaces Object with T but also uses the next field generically.

```
public class Babushka<T> {
  private T contents;
  private Babushka<T> next; // generic usage of a Babushka
}
```

Similarly, a method which takes and returns a Bubushka now uses the generic <T> so Babushka methods

```
public void insert(Babushka doll) { next = doll; }
public Babushka open() { return next; }
```

become the generic methods

```
public void insert(Babushka<T> doll) { next = doll; }
public Babushka<T> open() { return next; }
```

5 How to use an array of generic types

When creating an array of generic data types, you still have to call the array constructor using Object but you also have to cast to your generic type (T).

```
Object[] data = new Object[size]; // non-generic array
T[] data = (T[]) new Object[size]; // generic array
```

A common error with arrays is to call the constructor using your generic type (T).

```
T[] data = (T[]) new T[size];  // This is a common error!
```

From this point, you can use the array as normal. You should not need to cast anything when retrieving it from the array.