Ch 17 Generics

*A **generic** class or method is one that permits you to specify the allowable types of objects that the class or method may work with. (page 1005)*

Why bother with all this? Two primary advantages (examples on page 1006-1007):

- Basically, "safer" than casting Object
- Can catch some errors at compile time

Generics fixed big problems with Java's standard library:

> *Pre-Java 5 Downcasting from Object In versions before Java 5 a problem is that an object returned from a data structure must be downcast to whichever type you want. This was a constant annoyance, and was a form of weak-typing, which allowed run-time type mismatches that should be caught at compile time. Generic types, as in C++ templates, are in Java 5 to enforce typing and remove the need for explicit casting. For compatibility the older versions of the classes also work.*
> *- www.leepoint.net/notes-java/data/collections/05collections.html*

We've used ArrayList, so you know the new way (bottom of page 1007, too)
Here's the old way:

```
ArrayList names = new ArrayList();
names.add( "Prof Bill");
String s = (String) names.get( 0);   // cast Object return value
```

17.1 Intro

Heterogeneous object lists are "dangerous"... can lead to run-time errors, undetectable at compile-time.
Eliminate this danger by creating a generic class, where only one class of objects is handled and that class is explicitly specified.

17.2 Writing a generic class

Source: docs.oracle.com/javase/tutorial/java/generics/types.html
A generic class is defined with the following format:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

In <T>, T stands for "type". This is not a rule, just a convention. Here are some more:

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

Example (no generics):

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

Example (with generics):

```
public class Box<T> {
    private T obj;   // T stands for "Type"

    public void set(T obj) { this.obj = obj; }
    public T get() { return this.obj; }
}
```

With generics, the objects stored in Box are explicitly specified:

```
Box<Integer> integerBox;
Box<Weasel> boxOfWeasels = new Box<Weasel>();
```

In our Box of integers, you can use the primitive int.

```
int x = integerBox.get();    // unboxing
integerBox.set( 17);   // autoboxing
```

So, two definitions: **autoboxing** - auto-magically wrapping a primitive (like int) into an object (like Integer) when needed; and **unboxing** - the opposite of boxing

You can instantiate a generic class without a type. Then, `Object` is assumed. This is not recommended because you'll eventually have to cast and compiler can't check.
Example:

```
Box cassius = new Box();   // Object is used
```

17.3 Passing generic objects

You can specify generic type in method parameters. Example:

```
// theDeal is a list of strings
void groceryList( List<String> theDeal) { /* ... */ }
```

To leave the generic type unbounded, use the wildcard (?). Example:

```
// crate is a Box of any type, unrestricted
double shippingCost( Box<?> crate) { /* ... */ }
```

You can specify a class hierarchy "lower bound" using ? and `extends`. Example:

```
// arg is Box of Food or any subclass of Food
```

```
      void refrigerate( Box<? extends Food> arg) { /* ... */ }
```

You can specify a class hierarchy “upper bound” using ? and `super`. Example:

```
      // arg is Point of Integer or any super class of Integer
      void doIt( Point<? super Integer> arg) { /* ... */ }
```

17.4 Writing generic methods

You can use the generic type within a method you are writing. Example on page 1022 is good:

```
      public static <E> void displayArray( E[] array) {
         for( E element : array) {
            System.out.println( element);
         }
      }
```

17.5 Constraining type parameter in generic class

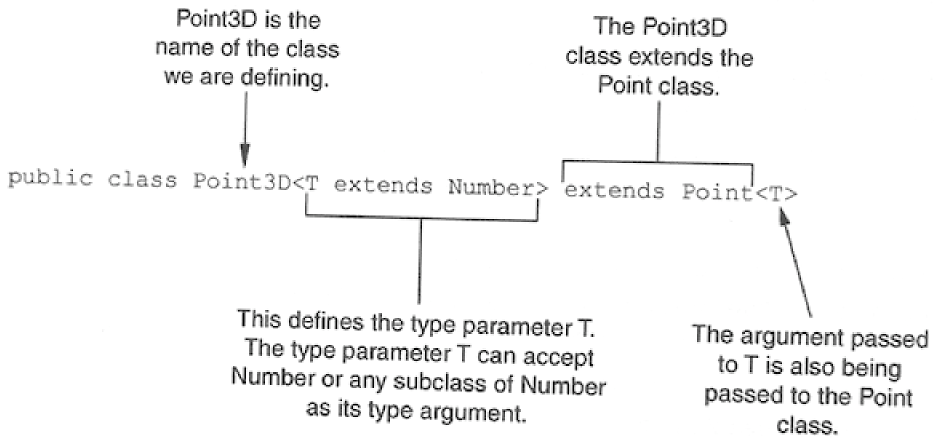
You can constrain generic type in your generic class. Book example:

```
      // Point of T which is Number or a subclass of Number
      public class Point<T extends Number> {
         /* … */
      }
```

17.6 Inheritance and generic classes

“A generic class can be a superclass, and it can extend other classes” (page 1026)

Book example (page 1028) explains this “difficult” syntax.


Point3D is the name of the class we are defining.
The Point3D class extends the Point class.
public class Point3D<T extends Number> extends Point<T>
This defines the type parameter T. The type parameter T can accept Number or any subclass of Number as its type argument.
The argument passed to T is also being passed to the Point class.

17.7 Multiple type parameters

In Collections, the Map interface uses two generic parameters.

: key and value.

```
// K is key; V is value
public interface Map<K,V> { /* … */ }
```

And you can create a Map like this:

```
// creates a HashMap with String key and values are a list of Strings
Map<String, List<String>> myMap = new HashMap<String, List<String>>();
```

17.8 Generics and interfaces

Generic interfaces work just like classes. Comparable!

```
public interface Comparable<T> {
   int compareTo( T obj);
}
```

17.9 Erasure

The punch line: generics are all an illusion. A compiler illusion. Sort of like a macro.

“When the Java compiler processes a generic class or method, it erases the generic notation and substitutes an actual type for each type parameter.” (page 1039)

This makes sense - we only use generics to get better compile-time checking anyway. Yes?

17.10 Restrictions on generics

Source: docs.oracle.com/javase/tutorial/java/generics/restrictions.html

1. You cannot create an instance of a type parameter. Example:

```
public static <E> void append(List<E> list) {
    E elem = new E();  // compile-time error
    list.add(elem);
}
```

2. You cannot create an array of generic objects

```
// compile-time error
List<Integer>[] arrayOfLists = new List<Integer>[2];
```

3. Generics can be static fields or referred to in a static method

4. You cannot make an exception class generic