What is Generics?

Generics is the capability to parameterize types. With this capability, you can define a class or a method with generic types that can be substituted using concrete types by the compiler. For example, you may define a generic stack class that stores the elements of a generic type. From this generic class, you may create a stack object for holding strings and a stack object for holding numbers. Here, strings and numbers are concrete types that replace the generic type.

Why Generics?

The key benefit of generics is to enable errors to be detected at compile time rather than at runtime. A generic class or method permits you to specify allowable types of objects that the class or method may work with. If you attempt to use the class or method with an incompatible object, a compile error occurs.

Generic Type

```
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5

```
package java.lang;

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

(b) JDK 1.5

Generic Instantiation

Runtime error

```
Comparable c = new Date();
System.out.println(c.compareTo("red"));
```

(a) Prior to JDK 1.5

```
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

(b) JDK 1.5

Improves reliability

Compile error

Generic ArrayList in JDK 1.5

java.util.ArrayList
+ArrayList() +add(o: Object): void +add(index: int, o: Object): void +clear(): void +contains(o: Object): boolean +get(index:int): Object +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: Object): Object

(a) ArrayList before JDK 1.5

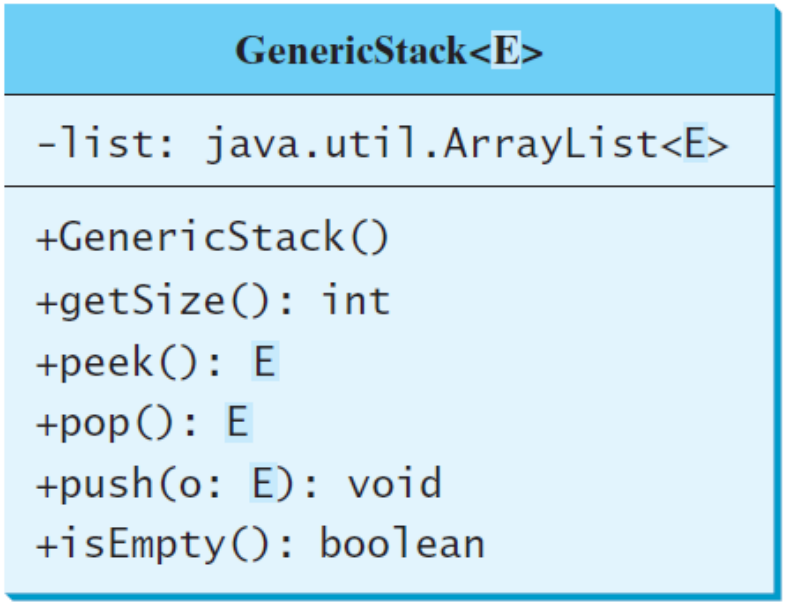
java.util.ArrayList<E>
+ArrayList() +add(o: E): void +add(index: int, o: E): void +clear(): void +contains(o: Object): boolean +get(index:int): E +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: E): E

(b) ArrayList since JDK 1.5

No Casting Needed

```
ArrayList<Double> list = new ArrayList<>();

list.add(5.5); // 5.5 is automatically converted to new Double(5.5)

list.add(3.0); // 3.0 is automatically converted to new Double(3.0)

Double doubleObject = list.get(0); // No casting is needed

double d = list.get(1); // Automatically converted to double
```

Declaring Generic Classes and Interfaces



An array list to store elements.	
Creates an empty stack.	
Returns the number of elements in this stack.	
Returns the top element in this stack.	
Returns and removes the top element in this stack.	
Adds a new element to the top of this stack.	
Returns true if the stack is empty.	

Generic Methods

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

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

Bounded Generic Type

```
public static void main(String[] args ) {
   Rectangle rectangle = new Rectangle(2, 2);
   Circle circle = new Circle (2);
   System.out.println("Same area? " + equalArea(rectangle, circle));
}

public static <E extends GeometricObject> boolean
      equalArea(E object1, E object2) {
   return object1.getArea() == object2.getArea();
}
```

Raw Type and Backward Compatibility

```
// raw type
ArrayList list = new ArrayList();
```

This is *roughly* equivalent to

```
ArrayList<Object> list = new ArrayList<Object>();
```

Raw Type is Unsafe

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum between two objects */
  public static Comparable max(Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

Runtime Error:

```
Max.max("Welcome", 23);
```

Make it Safe

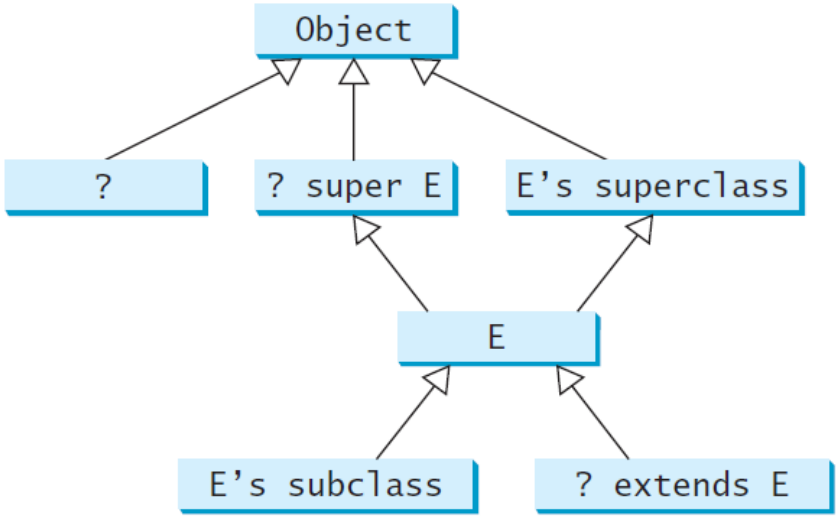
```
// Max1.java: Find a maximum object
public class Max1 {
  /** Return the maximum between two objects */
  public static <E extends Comparable<E>> E max(E o1, E o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}

Max.max("Welcome", 23);
```

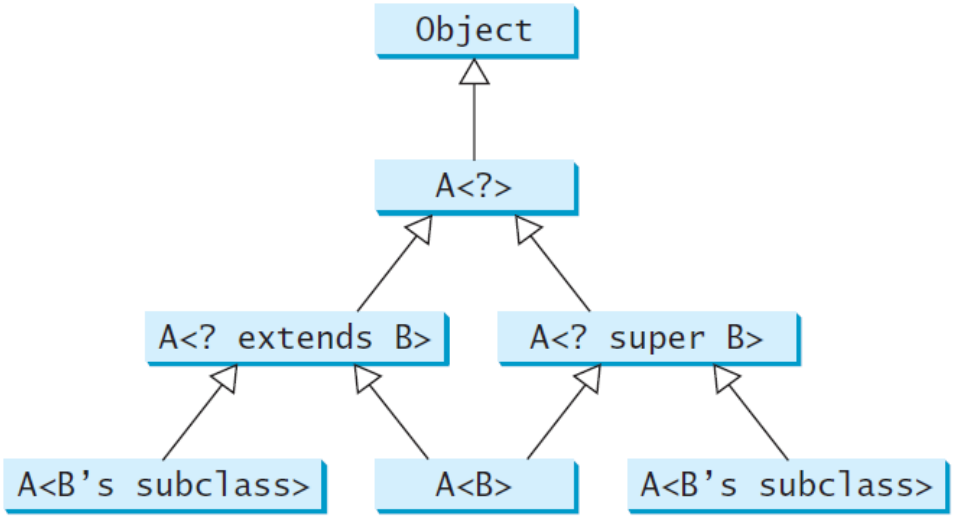
Wildcards

Why wildcards are necessary?

?	unbounded wildcard
? extends T	bounded wildcard
? super T	lower bound wildcard

Generic Types and Wildcard Types





Erasure and Restrictions on Generics

Generics are implemented using an approach called *type erasure*. The compiler uses the generic type information to compile the code, but erases it afterwards. So the generic information is not available at run time. This approach enables the generic code to be backward-compatible with the legacy code that uses raw types.

Compile Time Checking

For example, the compiler checks whether generics is used correctly for the following code in (a) and translates it into the equivalent code in (b) for runtime use. The code in (b) uses the raw type.

```
ArrayList<String> list = new ArrayList<>();
list.add("Oklahoma");
String state = list.get(0);
```

(a)

```
ArrayList list = new ArrayList();
list.add("Oklahoma");
String state = (String)(list.get(0));
```

(b)

Important Facts

It is important to note that a generic class is shared by all its instances regardless of its actual generic type.

```
GenericStack<String> stack1 = new GenericStack<>();
GenericStack<Integer> stack2 = new GenericStack<>();
```

Although GenericStack<String> and GenericStack<Integer> are two types, but there is only one class GenericStack loaded into the JVM.

Restrictions on Generics

- Restriction 1: Cannot Create an Instance of a Generic Type. (i.e., new E()).
- Restriction 2: Generic Array Creation is Not Allowed. (i.e., new E[100]).
- Restriction 3: A Generic Type Parameter of a Class Is Not Allowed in a Static Context.
- Restriction 4: Exception Classes Cannot be Generic.