**Generics :**

Generics was added in Java 5 to provide **compile-time type checking** and removing risk of ClassCastException that was common while working with collection classes.

The whole collection framework was re-written to use generics for type-safety.

We can define our own classes with generics type. A generic type is a class or interface that is parameterized over types

Generic Methods:

You can write **a single generic method declaration** that can be called with arguments of different types.

public class GenericMethodTest {

// generic method printArray

**public static < E > void printArray( E[] inputArray ) {**

**// Display array elements**

**for(E element : inputArray) {**

**System.out.printf("%s ", element);**

**}**

}

public static void main(String args[]) {

// Create arrays of Integer, Double and Character

Integer[] intArray = { 1, 2, 3, 4, 5 };

Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };

Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

System.out.println("Array integerArray contains:");

printArray(intArray); // pass an Integer array

System.out.println("\nArray doubleArray contains:");

printArray(doubleArray); // pass a Double array

System.out.println("\nArray characterArray contains:");

printArray(charArray); // pass a Character array

}

}

**generic class**

ublic class Box<T> {

private T t;

public void add(T t) {

this.t = t;

}

public T get() {

return t;

}

public static void main(String[] args) {

Box<Integer> integerBox = new Box<Integer>();

Box<String> stringBox = new Box<String>();

integerBox.add(new Integer(10));

stringBox.add(new String("Hello World"));

System.out.printf("Integer Value :%d\n\n", integerBox.get());

System.out.printf("String Value :%s\n", stringBox.get());

}

}

**Generics Terminology:**

For example, class Sender**<T>** indicates, the allowed reference type to create

instance of Sender are:

Any reference type T

Subclass of T

The **wildcard** ? is used to indicate **any type**.

There are two variations in using wildcard.

**? super T:** indicates lower bound meaning, any reference types which are superclass of T are allowed.

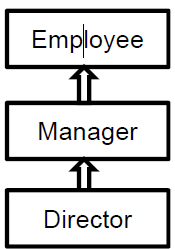
**? extends T:** indicated upper bound meaning, any reference types which are subclass of T are allowed.

Consider the given example for inheritance relationship.

**List<? super Manager>** means, list can be created of Manager, Employee etc. That is all

superclass's of Manager.

**List<? extends Manager>** means, list can be created of Manager, Director etc. That is all subclasses of Manager.



What are bounded and unbounded wildcards in Generics

*bounded and unbounded wildcards in generics* are used to bound any Type.

Bounded

Type can be upper bounded by using <? extends T> where **all Types must be sub-class of T** or lower bounded using <? super T> where **all Types required to be the super class of T**, here T represent the lower bound.

Unbounded

Single **<?> is called an unbounded wildcard in generic** and it can represent any type, similar to Object in Java.

For example  **List<?>** can represent any List e.g. List<String> or List<Integer> its provides highest level of flexibility on passing method argument.

on the other hand, **bounded wildcards provide limited flexibility within bound**. Any Type with bounded wildcards can only be instantiated within bound and any instantiation outside bound will result in compiler error.One of the important benefits of using bounded wildcard is that it not only **restrict the number of Types** can be passed to any method as an argument it also provides access to methods declared  by bound.

for example TreeMap(Comparator<? super K> comparator) allows access to compare() method of [Comparator in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html).

## Example of Bounded and Unbounded wildcards in Java Generics:

Java Collection frameworks have several *examples of using bounded and unbounded wildcards in generics*.

Utility method provided in Collections class accepts parameterized arguments. Collections.unmodifiableSet(Set<? extends T> s) and Collections.unmodifiableMap(Map<? extends K,? extends V> m) are written using  bounded wildcards which allow them to operate on either Collection of T or Collection of subclass or superclass of T. just look at Java API for 1.5 and you will find lot of example of bounded and unbounded generic wildcards within JDK itself. If you are learning Java 1.5  you can also check [my post on Java Enum](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html) and  [Variable arguments in Java](http://javarevisited.blogspot.com/2011/09/variable-argument-in-java5-varargs.html).

### When to use super and extends wildcards in Generics Java

Since there are two kinds of bounded wildcards in generics, **super and extends**, When should you use the super wildcard and when should you extends wildcards. Joshua Bloch in Effective Java book has suggested **Producer extends, Consumer super** mnemonic regarding the use of bounded wildcards. This book also has some good advice regarding how to use generics in Java and if you haven’t read it already, it's worth reading the book for Java programmer. Anyway if type T is used as a producer than use <? extends T>  and if type T represent consumer than use <? super T> bounded wildcards. Bounded wildcards in generics also increase the flexibility of any API. To me its question of requirement, if a method also needs to accept any implementation of T then use extends wildcards.

### Difference between ArrayList<? extends T>  and ArrayList<? super T>

This is one of popular **generics interview question** , which is asked to check whether you are familiar to **bounded wildcards in generics**. both <? extends T> and <? super T> represent bounded wildcards, one will accept only T or subclass while other will accept T or super class. *bounded wildcards* give more flexibility to methods which can operate on a collection of T or its subclass. If you look at java.util.Collections class you will find several example of bounded wildcards in generics method. e.g. Collections.unmodifiableSet(Set<? extends T> s) will accept Set of type T or Set of subclass of T.

That's all on **what is bounded wildcards in generics**.

both bounded and unbounded wildcards provide a lot of flexibility on API design especially because Generics is not covariant and List<String> can not be used in place of List<Object>. Bounded wildcards allow you to write methods which can operate on Collection of Type as well as Collection of Type subclasses.

Usage of Generics:

Generic classes in use can have multiple arguments.

Arguments can be standard as well as custom types

**Note: Generic classes cannot be assigned according to the super or subclass hierarchy of them.**

**Note: Generic classes do not support inheritance relationship between type arguments.**

**Note: Although the inheritance relationship between the type arguments of the generic classes does not exist, Inheritance relationship between Generic classes themselves still exist.**

**The most commonly used type parameter names are:**

* **E – Element (used extensively by the Java Collections Framework, for example ArrayList, Set etc.)**
* **K – Key (Used in Map)**
* **N – Number**
* **T – Type**
* **V – Value (Used in Map)**
* **S,U,V etc. – 2nd, 3rd, 4th types**