

Generics and Collections





Review: Generics

With generics, we can define a class that takes a **type as a parameter**

Ex: List <Integer>, Map<String>



Review: Type erasure

Once your code is compiled, the generic types are **erased**.



Pre-compilation vs post compilation

```
public static void typeErasureExample() {  
    List<String> a = new ArrayList<String>();  
    a.add("foo");  
    String foo = a.get(0);  
}
```

```
public static void typeErasureExample() {  
    List<String> a = new ArrayList();  
    a.add("foo");  
    String foo = (String)a.get(0);  
}
```



Arrays of Generics

```
class Example {  
    public static T[] makeArray(int length) {  
        return new T[length];           //Illegal!  
    }  
}
```



Solutions

```
class Example {  
    public static <T> T[] makeArray(int length) {  
        return (T[]) (new Object [length]);    //Legal, but not always safe  
    }  
}
```

This array can never be assigned to a variable with a concrete (non-generic) type.



Solutions

```
class Example {  
    public static <T> T[] makeArray(int length) {  
        return (T[]) (new Object [length]);  
    }  
  
    public static String[] makeStringArray(int length) {  
        String[] s = Example.<String>makeArray(length); //Not ok  
        return s;  
    }  
  
    public static <T> void printLength(int length) {  
        T[] s = <T>makeArray(length); //OK!  
        System.out.println(s.length);  
    }  
}
```



Solutions

Use a Collection instead of an array!



Review: Wildcards

Recall that `List<String>` is **not** a subtype of `List<Object>`, even though `String` **is** a subtype of `Object`.

How do we program for cases covering related types?



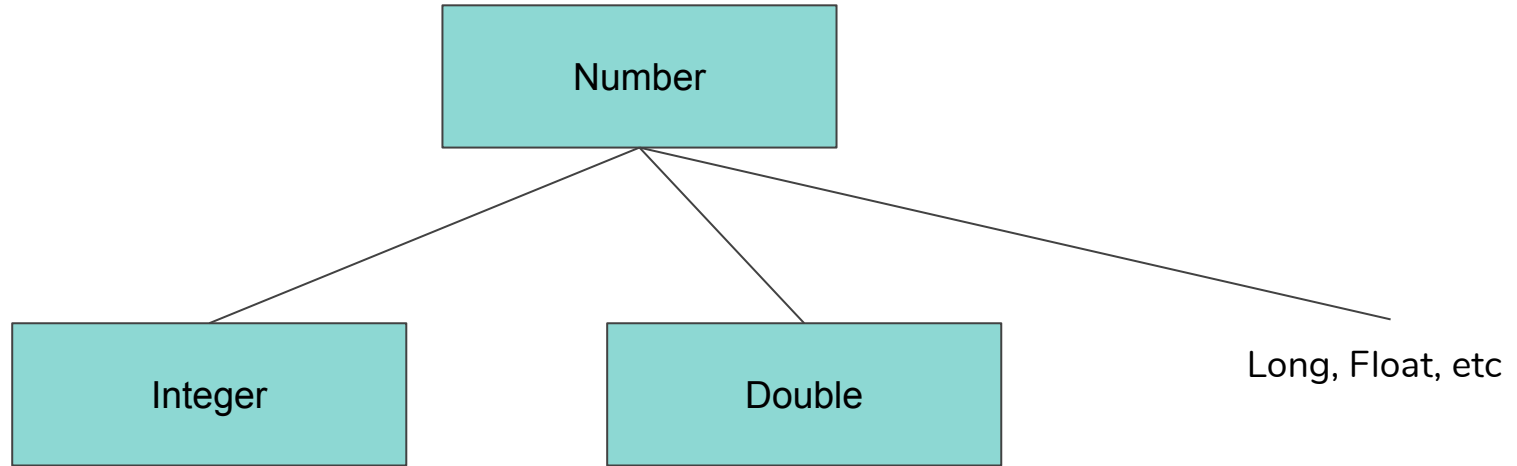
Review: Wildcards

There are three wildcard operators we can use to represent groups of types in generics.

- List `<?>`
 - A List of any type
- List `<? extends A>`
 - A List of any subtype of A
- List `<? super A>`
 - A List of any supertype of A



Type Hierarchy of Number, Integer, Double





Wildcards

```
List<?> unbound = new ArrayList<>();  
List<? extends Number> ext = new ArrayList<>();  
List<? super Number> sup = new ArrayList<>();
```

```
unbound.add(0);           //Illegal  
ext.add(0);               //Illegal  
sup.add(0);              //Legal
```



Why this works the way it does

```
List<String> strList = new ArrayList<>();  
strList.add("hello");  
List<?> unbound = strList;  
Integer zero = 0;
```

```
unbound.add(zero); //Illegal! This would be adding an Integer to a String list
```



Why this works the way it does

```
List<Double> dbList = new ArrayList<>();  
dbList.add(2.112);  
List<? extends Number> ext = dbList;  
Integer zero = 0;
```

```
ext.add(zero); //Illegal! This would be adding an Integer to a Double list
```



Why this works the way it does

```
List<Object> oList = new ArrayList<>();  
oList.add("hello");  
List<? super Number> sup = oList;  
Integer zero = 0;
```

```
sup.add(zero); //Legal zero is still an Object, which is a super class of Number
```



Wildcards

```
List<Integer> original= new ArrayList<>();  
original.add(0);
```

```
List<?> unbound = original;  
List<? extends Integer> ext = original;  
List<? super Integer> sup = original;
```

```
Integer i = unbound.get(0); //Illegal  
Integer i = ext.get(0);    //Legal  
Integer i = sup.get(0);    //Illegal
```




Wildcards

```
List<String> original= new ArrayList<>();  
original.add("hello");
```

```
List<?> unbound = original;
```

```
Integer i = unbound.get(0);    //Illegal, we have no clue what type is inside of unbound
```



Wildcards

```
List<Integer> original= new ArrayList<>();  
original.add(0);
```

```
List<? extends Integer> ext = original;
```

```
Integer i = ext.get(0);    //Legal, regardless of the type of the list, we know it is a subtype  
                           // of Integer!
```



Wildcards

```
List<Number> original= new ArrayList<>();  
original.add(2.112);
```

```
List<? super Integer> sup = original;
```

```
Integer i = sup.get(0);    //Illegal, we only know that the list is a super type of Integer. It  
                           //could be a Number list or even an Object list.
```



Raw Types

You don't actually have to specify a type!

```
List a = new ArrayList<>();    //this is dangerous, java will warn you
a.add(2112);                  //Legal
a.add("hello");               //Legal
a.add(true);                  //Legal
a.add(null);                  //Legal
```

Only available to maintain backwards compatibility with older versions of Java



Solutions

```
import java.lang.reflect.Array;
class Example <T> {
    public static <T> T[] makeArray(Class<T> clazz, int length) {
        return (T[]) Array.newInstance(clazz, length);    //Legal and safe, but gross.
    }
}
```

Unless you have an extremely good reason, please do not do this. You may lose points on your A3 and future assignments.



Collection

Interface that describes collections of items.

Ex: ArrayList, LinkedList, HashSet

Maps act similarly to Collections although they are not a subtype, so people often refer to Map as part of the “Collections Framework”



Useful Methods in Collection

boolean add(E e) - adds an element of type E, returns whether the collection added the element

boolean contains(Object o) - returns true if o is in the collection

boolean remove(Object o) - removes the element equal to o if it is present, returns whether something was actually removed

int size() - returns the number of elements in the collection



Iterating

You can also iterate through elements of a collection with a foreach loop

Ex:

```
ArrayList<Integer> nums = ....;
```

```
for (Integer i : nums) {  
    //do stuff  
}
```




Array of Collection

This is probably the only case in which you would want to use a raw type!

```
ArrayList<Integer>[] arrListArr = new ArrayList<Integer>[10];           // Illegal  
ArrayList<Integer>[] arrListArr2 = new ArrayList[10];                  // Legal
```



Is this Legal?

```
Set<?> setOfUnknownType = new LinkedHashSet<String>();  
setOfUnknownType = new LinkedHashSet<Integer>();
```



Is this Legal?

```
Set<String> setOfString = new HashSet<String>();  
Set<Object> SetOfObject = new HashSet<String>();
```



Is this Legal?

```
Set<? extends Number> set = new HashSet<Integer>();  
set = new HashSet<Float>();
```



Will this Compile?

```
public static void print(List<? extends Number> list) {  
    for (Number n : list) {  
        System.out.print(n + " ");  
    }  
    System.out.println();  
}
```