SEMINAR – GENERICS

A simple non generic code and the problem

```
public class SingleBox {
   private Object first;
   public void setFirst(Object object) { this.first = object; }
   public Object getFirst() { return first; }
}
```

a main method to use the previous class:

```
SingleBox ob= new SingleBox();
Integer i = new Integer(10);
ob.setFirst(i);//information about Integer is lost
//we need a downcast to recover the specific information, namely the Integer
Integer a = (Integer) ob.getFirst();
//the cast may be wrong, it will pass the compiler but a runtime error will occur
String s = (String) ob.getFirst();
```

Generic code

a simple example to show how to generalize the code:

```
public class SingleBox<T> {
   // T stands for "Type"
   protected T first;
   public void setFirst(T t) { this.first = t; }
   public T getFirst() { return first; }
}
```

a main method:

```
SingleBox<Integer> ob= new SingleBox<Integer>();
Integer i = new Integer(10);
ob.setFirst(i);
Integer a = ob.getFirst();
//an error will be generated at compile time
String s = ob.getFirst();
```

An inheritance example

```
public class PairBox<T1, T2> extends SingleBox<T1> {
   private T2 second;
   public PairBox(T1 f, T2 s) {
         this.first = f;
         this.second = s;
   }

   public T2 getSecond()       { return second; }
   public void setSecond(T2 t) { this.second=t; }
}
```

An example of raw types

```
SingleBox<String> stringBox = new SingleBox<>();
SingleBox rawBox = stringBox;               // OK
```

But if you assign a raw type to a parameterized type, you get a warning:

```
SingleBox rawBox = new SingleBox();          // rawBox is a raw type of Box<T>
SingleBox<Integer> intBox = rawBox;     // warning: unchecked conversion
```

You also get a warning if you use a raw type to invoke generic
methods defined in the corresponding generic type:

```
SingleBox<String> stringBox = new SingleBox<>();
SingleBox rawBox = stringBox;
rawBox.setFist(new Integer(8));  // warning: unchecked invocation to set(T)
```

The warning shows that raw types bypass generic type checks,
deferring the catch of unsafe code to runtime. Therefore, you should avoid using raw types.

An example of bounded type parameters

```
public class NaturalNumber<T extends Integer> {
   private T n;
   public NaturalNumber(T n)  { this.n = n; }
   public boolean isEven() {
      return n.intValue() % 2 == 0;
//T must be a subclass of Integer. Otherwise the compiler will signal an error
   }

   // ...
}
```

A generic algorithm

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

A generic algorithm to compare the first fields:

```
public static <T extends Comparable<T>>
int checkFields(SingleBox<T> box, T elem) {
       if (elem.compareTo(box.getFirst()) >= 0)
            return 1;
       else
         return -1;

}

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

A generic algorithm to compare the first fields:

```
public static <T extends Comparable>
int checkFields(SingleBox<T1> box, T elem) {
       if (elem.compareTo(box.getFirst()) >= 0)
            return 1;
       else
         return -1;
```

Discuss the diference between Comparable and Comparable<T>.

Motivation for Wildcards

```
SingleBox<Number> box = new SingleBox<Number>();
box.setFirst(new Integer(10));   // OK
box.setFirst(new Double(10.1));  // OK

public void boxTest(SingleBox<Number> n) { /* ... */ }
```

Are you allowed to pass in SingleBox<Integer> or SingleBox<Double>, as you might expect? The answer is "no", because SingleBox<Integer> and SingleBox<Double> are not subtypes of SingleBox<Number>.

If we assume that SingleBox<Integer> is subtype of SingeBox<Number> we can have the followings:

```
SingleBox<Number> box = new SingleBox<Integer>(); //the assumption
box.setFirst(new Double(10.2)); //ERROR at runtime when you write the box content
```

Wildcards

SingleBox<?>

-the only allowed operation is to read Object and to write null.

- can be interpreted to have Object as content but you cannot write anything only null
- is the top of the hierarchy

```
SingleBox<?> wildbox= new SingleBox<Number>;
SingleBox<Integer> intbox= new SingleBox<Integer>;
wildbox = intbox; //OK
intbox.setFirst(new Integer(2)); //OK
wildbox.setFirst(new Integer(2)); //ERROR
Integer = intbox.getFirst(); //OK
Integer = wildbox.getFirst(); //ERROR
Object = wildbox.getFirst(); //OK
```

in fact SingleBox<?> means SingleBox<T> where bottom < T<Object

Bounded Wildcards

Upper Bound – we can only read the content—Covariant subtyping
==========
SingleBox<? extends Integer> a = new SingleBox<Integer>();//OK
SingleBox<? extends Number> b = a; // OK.
SingleBox<? extends Number> b =new SingleBox<Integer>();//OK
SingleBox<? extends Integer> a = new SingleBox<Number>();//ERROR

Upper bound means that we can only read elements of the type (or of superclass of the type) given by the upper bound.

SingleBox<? extends Number> means SingleBox<T> where bottom<T<Number
and we can read Number or a superclass of Number.

an example:

public Number processReadBox(Box<? extends Number> a) { return a.getFirst(); }

and can be called by proccessReadBox(v) where v has the type Box<? extends Integer>,
Integer is derived from Number or v can have the type Box<Integer>

Lower Bound – we can only write the content –contravariant subtyping
==========

SingleBox<? super Number> a = new SingleBox<Number>();//OK
SingleBox<? super Integer> b = a; // OK.
SingleBox<? super Number> b =new SingleBox<Integer>();//ERROR
SingleBox<? super Integer> a = new SingleBox<Number>();//OK

Lower bound means that we can write elements of type (or of subclasses of the type) given by the lower bound.

SingleBox<? super Integer> means SingleBox<T> where Integer<T<Object
and we can write Integer or a subclass of Integer.

an example:

public static void processWriteBox(Box<? super Integer> a, Integer x) { a.setFirst(x); }

and can be called by proccessWriteBox(v) where v has the type Box<? super Number>,
Integer is derived from Number or v can have the type Box<Integer>

SingleBox<?> wildbox;
SingleBox<? extends Number> nrbox;
SingleBox<? super Integer> intbox;
wildbox=nrbox; //OK
wildbox=intbox;//OK