Victoria University of Wellington, New Zealand
Te Whare Wananga o te Upoko o te Ika a Maui Aotearoa

SWEN221: Software Development

15: Generics II

David J. Pearce & Nicholas Cameron & James Noble & Marco Servetto
Engineering and Computer Science, Victoria University

Generics + Subtyping

A:yes
B: no

Is this true?

ArrayList<String> ≤ List<String>

Note: is equivalent to ask if the following compiles:

```
List<String> l = new ArrayList<String>();
```

Generics + Subtyping

```
String concat(List<String> ss) {
  String result = "";
  for (String s : ss) { result = result + s; }
  return result;
}
void doStuff() {
  ArrayList<String> ss = new ArrayList<String>();
  /*... do stuff ...*/
  System.out.println(concat(ss));
}
```

- Q) Does this work?

A:yes
B: no

Generics + Subtyping

`List<String>` $\leq$ `List<Object>` ?

Generics + Subtyping

```
void doSomething(List<Object> os) {

  /* ... */
}
void foo() {
  List<Object> os = new ArrayList<Object>();
  List<String> ss = new ArrayList<String>();
  /* ... */
  doSomething(os);//(1)is it ok?

  doSomething(ss);//(2)is it ok?
}
```

A:yes
B: no

A:yes
B: no

Generics + Subtyping

```
void doSomething(List<Object> os) {
  //write down an example of code
  /* ... */
}
void foo() {
  List<Object> os = new ArrayList<Object>();
  List<String> ss = new ArrayList<String>();
  /* ... */
  doSomething(os);//(1)is it ok?

  doSomething(ss);//(2)is it ok?
}
```

Generics + Subtyping

```
void doSomething(List<Object> os) {
  //for example, add something to os
  os.add(new Integer(1));
}
void foo() {
  List<Object> os = new ArrayList<Object>();
  List<String> ss = new ArrayList<String>();
  /* ... */
  doSomething(os);//(1)is it ok?

  doSomething(ss);//(2)is it ok?
}
```

Quiz – which compiles?

A

```
void print(List<Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

B

```
void print(List<String> ss) {
  for (String s : ss) { System.out.println(s);}
}
void foo() {
  List<Object> y = new ArrayList<Object>();
  print(y);}
```

C both wrong

Java Type

```
class Cup {
  Object f;
  Cup(Object f) {
    this.f = f;
  }
}
```

Cup

```
Cup myCup = new Cup(null);
```

Java Generics Type

```
class Cup<T> {
   T f;
   Cup(T f) {
      this.f = f;
   }
}
```

Cup<Tea>

```
Cup<Tea> myCup = new Cup<Tea>(new Tea());
```

Java Wildcards Type

```
class Cup<T> {
   T f;
   Cup(T f) {
      this.f = f;
   }
}
```

Cup<?>

```
Cup<?> myCup  = CupProvider.getCup();
```

Bounds

```
class Cup<T> {
   T f;
   Cup(T f) {
      this.f = f;
   }
}
```

```
Cup<? extends Drink>
```

```
Cup<? extends Drink> myCup = CupProvider.getCup();
```

Generics - Invariant Subtyping

$$\texttt{Cup<Tea>} \not\leq \texttt{Cup<Object>}$$

```
void putCandy(Cup<Object> c){ c.f=new Candy();}
```

```
Cup<Tea> ct=new Cup<Tea>(new Tea());
assert ct.f instanceof Tea;
putCandy(ct);//wrong method call, not compile
assert ct.f instanceof Tea;//if former call was
//accepted, now this would be a wrong assertion
```

Bounds

```
Cup<? extends Drink>
```

Generics - Invariant Subtyping

$$\texttt{Cup<Tea>} \not\leq \texttt{Cup<Object>}$$

Wildcards-Bounds-Variant Subtyping

$$\texttt{Cup<Tea>} \not\leq \texttt{Cup<Object>}$$

$$\texttt{Cup<Tea>} \leq \texttt{Cup<?>}$$

$$\texttt{Cup<Tea>} \leq \texttt{Cup<? extends Drink>}$$

$$\texttt{Cup<? extends Tea>} \leq \texttt{Cup<? extends Drink>}$$

Using variant subtyping for good

```
void print(List<Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

ok?
A/B
for
yes/no

Using variant subtyping for good

```
void print(List<Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

ok?
A/B
for
yes/no

```
void print(List<? extends Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

ok?
A/B
for
yes/no

Using variant subtyping for good

```
void print(List<Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

Wrong

```
void print(List<? extends Object> os) {
  for (Object o : os) { System.out.println(o);}
}
void foo() {
  List<String> y = new ArrayList<String>();
  print(y);}
```

Correct

Wildcard Capture

```
void m(Cup<?> c) { //here I have no idea
  this.test(c);    //what ? is
  }

<X> void test(Cup<X> cx) {
  //here I know that is a cup of X
  }
void swap(ArrayList<?> list){/*can I do it?*/}
//swap the first element with the second one
//try to write down a solution
```

Wildcard Capture

```
void m(Cup<?> c) { //here I have no idea
  this.test(c);    //what ? is
  }

<X> void test(Cup<X> cx) {
  //here I know that is a cup of X
  }
void swap(ArrayList<?> list){ swapAux(list);}
<X>void swapAux(ArrayList<X> list){



}
```

Wildcard Capture

```
void m(Cup<?> c) { //here I have no idea
  this.test(c);    //what ? is
  }

<X> void test(Cup<X> cx) {
  //here I know that is a cup of X
  }
void swap(ArrayList<?> list){ swapAux(list);}
<X>void swapAux(ArrayList<X> list){
  X e=list.get(0);
  list.set(0,list.get(1));
  list.set(1,e);
}
```

Wildcard Types

- Wildcard Types
  - Are indicated by a "?"
  - E.g. List<?> x
  - Not found in many other languages (e.g. not in C++ templates)

- What are they?
  - They are types, *but we don't know which they are*
    - E.g. List<?> could be a List<String> ...
    - Or, List<?> could be a List<Integer> ...
  - The point is: *we don't know which it is!*

Wildcard Types

- Subtype of all Lists is: `List<?>`

```
void print(List<?> xs) {
  for(Object x : xs) System.out.println(x);
}
void foo(){
 ArrayList<String> y = new ArrayList<String>();
 ArrayList<Integer> z = new ArrayList<Integer>();
 /*...*/
 print(y);
 print(z);
}
```

Both are OK

Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
  void print(List<Point> ps) {
   for(Point p : ps) {
     System.out.println("x=" + p.x + ", y=" + p.y);
    }}
  void foo(){
    ArrayList<Point> vp = new ArrayList<Point>();
    ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
    /*...*/
    print(vp);
    print(vcp);
  }}
```

OK? (A/B) for Yes/No

Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
  void print(List<?> ps) {
   for(Point p : ps) {
     System.out.println("x=" + p.x + ", y=" + p.y);
    }}
  void foo(){
    ArrayList<Point> vp = new ArrayList<Point>();
    ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
    /*...*/
    print(vp);
    print(vcp);
  }}
```

OK? (A/B) for Yes/No

Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
  void print(List<? extends Point> ps) {
   for(Point p : ps) {
     System.out.println("x=" + p.x + ", y=" + p.y);
    }}
  void foo(){
    ArrayList<Point> vp = new ArrayList<Point>();
    ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
    /*...*/
    print(vp);
    print(vcp);
  }}
```

OK? (A/B) for Yes/No

Exercise

```
void doSomething(List<? extends Point> ps) {
  ps.add(new Point(0,0));          // 1
  Object o = ps.get(0);            // 2
  Point p = ps.get(0);             // 3
}
void foo(){
  doSomething(new ArrayList<Point>());          //4
  doSomething(new ArrayList<ColoredPoint>());//5
}
```

- Q) Which statements are OK ?

  A/B for ok/not ok

  1) 2) 3) 4) 5)

Lower Bound

Rarely used, example in `java.util.Collections`

```
static <T> void sort(List<T> list, Comparator<? super T> c)
```

Sorts the specified list according to the order induced by the specified comparator.

- Here, "super" indicates a **lower bound**
  - i.e. a comparator able to compare any kind of super elements with respect to T is accepted
  - for example, a list of `ColoredPoints` can be compared with a `Comparator<Point>`.