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

SWEN221: Software Development

18: Generics II

Yi Mei
Engineering and Computer Science, Victoria University
Modified from David J. Pearce & Nicholas Cameron & James Noble

Generics + Subtyping

Is `ArrayList<String>` **a subtype of** `List<String>`**?**

$$\texttt{ArrayList<String>} \leq \texttt{List<String>}?$$

Generics + Subtyping

```
int countElements(List<String> v) {
 int count=0;
 for(Object o : v) { count = count+1; }
 return count;
}

ArrayList<String> v = new ArrayList<String>();
…
int total = countElements(v);
```

- Q) Does this work?

Generics + Subtyping

```
int countElements(List<String> v) {
 int count=0;
 for(Object o : v) { count = count+1; }
 return count;
}

ArrayList<String> v = new ArrayList<String>();
…
int total = countElements(v);
```

- Q) Does this work? ✔

So, `ArrayList<String>` ≤ `List<String>`

Generics + Subtyping

```
List<String> ≤ List<Object> ?
```

Generics + Subtyping

```
void doSomething(List<Object> v) {
 v.add(new Integer(1));
}

List<Object> vo = new ArrayList<Object>();
List<String> vc = new ArrayList<String>();
…
doSomething(vo);
doSomething(vc);
```

- Q) Does this work?

Generics + Subtyping

```
void doSomething(List<Object> v) {
 v.add(new Integer(1));
}

List<Object> vo = new ArrayList<Object>();
List<String> vc = new ArrayList<String>();
…
doSomething(vo);
doSomething(vc);
```

- Q) Does this work? ✗

So, `List<String>` $\not\leq$ `List<Object>`

Generics + Subtyping

- `MyClass<A>` has **NO** relationship with `MyClass<B>`, no matter whether `A` and `B` are related or not

Quiz – which compiles?

```
void print(List<Object> xs) {
 for(Object x : xs) System.out.println(x);
}

List<String> y = new ArrayList<String>();
print(y);
```

1

```
void print(List<String> xs) {
 for(String x : xs) System.out.println(x);
}

List<Object> y = new ArrayList<Object>();
print(y);
```

2

A) 1 B) 2 C) neither

Quiz – which compiles?

```
void print(List<Object> xs) {
 for(Object x : xs) System.out.println(x);
}

List<String> y = new ArrayList<String>();
print(y);
```

1

```
void print(List<String> xs) {
 for(String x : xs) System.out.println(x);
}

List<Object> y = new ArrayList<Object>();
print(y);
```

2

A) 1 ✗ B) 2 ✗ C) neither ✓

Wildcard Types

- Can we create relationships between generic classes
  - `MyClass<A>`
  - `MyClass<B>`
- when `A` and `B` are related?

Wildcard Types

- Wildcard Types
  - Are indicated by a “?”
  - E.g. List<?> x

- What are they?
  - They are *anonymous types*
  - 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!*

- **NOT in generic type definition**

Wildcard Types

```
class Cup<T> {
 public T content;
 Cup(T c) {
  content = c;
 }}
```

```
Cup<?> cup = getCup();
```

subtype

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

```
Cup<Coffee> cup =
  new Cup<Coffee>(new Coffee());
```

Wildcard Types

- `Cup<?>`: subtype of all Cups

```
void drink(Cup<?> c) {
 System.out.println("Drink a cup of " +
   c.content.toString();
}

Cup<Tea> c1 = new Cup<Tea>(new Tea());
Cup<Coffee> c2 = new Cup<Coffee>(new Coffee());

drink(c1);
drink(c2);
```

- OK or not?

Wildcard Types

- `Cup<?>`: subtype of all Cups

```
void drink(Cup<?> c) {
 System.out.println("Drink a cup of " +
   c.content.toString();
}

Cup<Tea> c1 = new Cup<Tea>(new Tea());
Cup<Coffee> c2 = new Cup<Coffee>(new Coffee());

drink(c1);
drink(c2);
```

Both are OK

- OK or not?

Wildcard Types

```
void drink(Cup<?> c) {
 c.content.drink();
 System.out.println("Drink a cup of " +
   c.content.toString();
}

Cup<Tea> c1 = new Cup<Tea>(new Tea());
Cup<Coffee> c2 = new Cup<Coffee>(new Coffee());


drink(c1);
drink(c2);
```

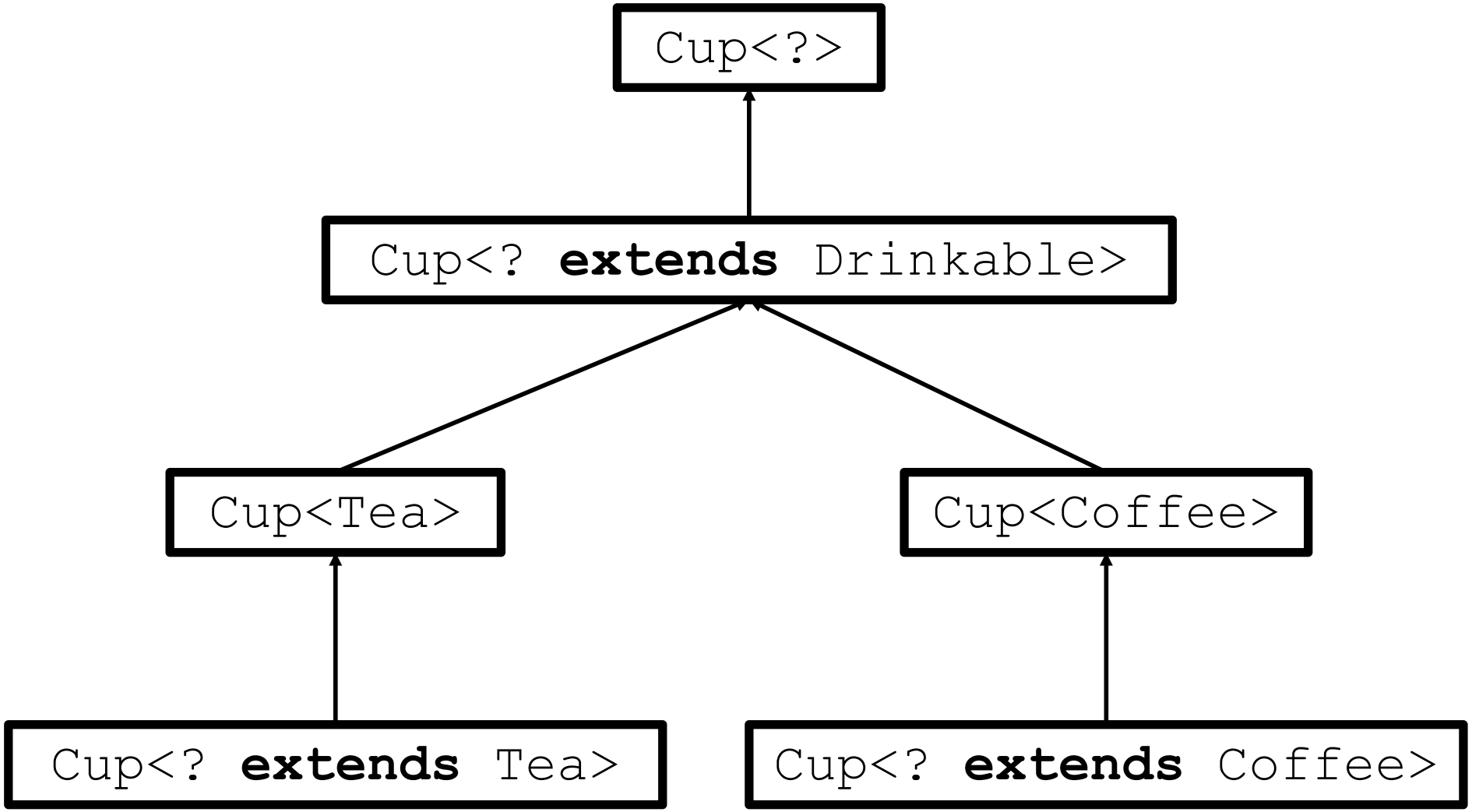
- OK or not?

Wildcards + Type Bounds

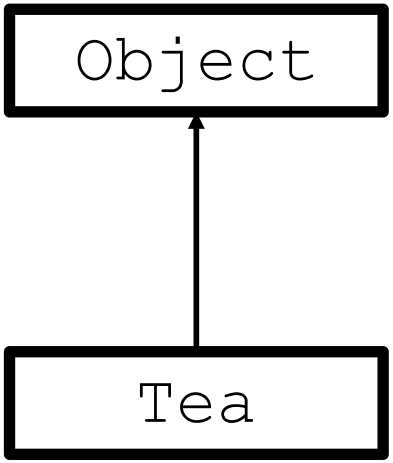
```
void drink(Cup<? extends Drinkable> c) {
 c.content.drink();
 System.out.println("Drink a cup of " +
   c.content.toString();
}

Cup<Tea> c1 = new Cup<Tea>(new Tea());
Cup<Coffee> c2 = new Cup<Coffee>(new Coffee());


drink(c1);
drink(c2);
```

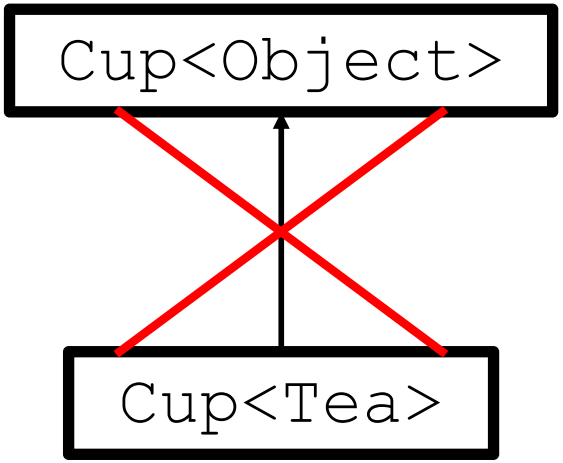
```
interface Drinkable { void drink(); }
class Tea implements Drinkable { … }
class Coffee implements Drinkable { … }
```

Wildcards + Type Bounds



Wildcards + Type Bounds





Quiz

A

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

B

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

- Q) Which are working?

A **B** **Both** **None**

Quiz

A

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

B

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

- Q) Which are working?

A ✗ B ✓ Both ✗ None ✗

Wildcard Capture

```
void foo(List<?> x) {
 x.set(0, x.get(0));
}
```

This is an `Object`

Cannot confirm what type of Object to set

```
void foo(List<?> x) {
 fooHelper(x);
}
// Helper method created so that the wildcard can be captured
// through type inference.
<T> void fooHelper(List<T> x) {
 x.set(0, x.get(0));
}
```

Upper Bound Example

```
class Point { int x; int y; … }
class ColPoint extends Point { int colour; }

class PointGroup<T extends Point> {
 private List<T> points = …;

 public void add(Point p) {
  points.add(p);
 }
}
```

- What's wrong with this?

Lower Bound Example

```
class Point { int x; int y; … }
class ColPoint extends Point { int colour; }

class PointGroup<T extends Point> {
 private List<T> points = …;

 public void write(List<? super Point> out) {
  out.addAll(points);
 }
}
```

- Here, "super" indicates a **lower bound**
  - i.e. **Cannot** be subtype of Point!
  - Why is this useful?

Summary

- Generics syntax
- Upper/Lower bounds for generic type
- Generic classes/methods
- Type Erasure
- Generics subtyping
- Wildcards + type bounds