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

SWEN221: Software Development

17: Generics I

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

Why Generics?

```
class Vec {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(Object e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public Object get(int index) {
  if(index >= end) { throw … }
  else return elems[index];
}}
```

```
java: incompatible types:
java.lang.Object cannot be converted to generics.Cat
```

```
Vec v = new Vec();
v.add(new Cat());
Cat c = v.get(0);
```

Why Generics?

```
class Vec {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(Object e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public Object get(int index) {
  if(index >= end) { throw … }
  else return elems[index];
}}
```

```
Vec v = new Vec();
v.add(new Cat());
Cat c = (Cat) v.get(0); // have to cast :-(
```

This says v is a Vec of Objects

We know this returns a Cat, but we still have to cast

How can we say v is a Vec of Cats?

Java Generics

- History
  - Introduced in Java 1.5
  - Similar to C++ templates, but actually quite different as well!

- Before Java generics:
  - **Can only say things like: 'v' is a Vector of Objects**
  - Then, can put any Object into 'v' without restriction
  - With a Vector of just Cats, have to cast Objects to Cats

- With Java Generics:
  - **Can say things like: 'v' is a Vector of Cats**
  - Then, can only put Cats into 'v'
  - And, can only get Cats out of 'v' – no casting required!

The Vec Class

```
class Vec {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(Object e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public Object get(int index) {
  if(index >= end) { throw … }
  else return elems[index];
}}
```

```
Vec v = new Vec();
v.add(new Cat());
Cat c = (Cat) v.get(0); // have to cast :-(
```

The Generic Vec Class

```
class Vec<T> {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(T e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public T get(int index) {
  if(index >= end) { throw … }
  else return (T) elems[index];
}}
```

```
Vec<Cat> v = new Vec<Cat>();
v.add(new Cat());
Cat c = v.get(0); // don't have to cast :-)
```

"T" is a generic parameter

"T" represents the type of object held in Vec

This says v is a Vec of Cats

Can only put Cats into v

Can only get Cats out of v

Pair Example

```
class Pair {
 private Object first;
 private Object second;

 public Pair(Object f, Object s) {
  first = f; second = s;
 }
 public Object first() { return first; }
 public Object second() { return second; }
}
```

```
Pair p1 = new Pair("Cat",1);
Pair p2 = new Pair(10,20);
String c = (String) p1.first();
Integer i = (Integer) p2.first();
```

Pair Example

```
class Pair<FIRST,SECOND> {
 private FIRST first;
 private SECOND second;

 public Pair(FIRST f, SECOND s) {
  first = f; second = s;
 }
 public FIRST first() { return first; }
 public SECOND second() { return second; }
}
```

No need to cast!

```
Pair<String,Integer> p1 = new Pair<String,Integer>("Cat",1);
Pair<Integer,Integer> p2 = new Pair<Integer,Integer>(10,20);
String c = p1.first();
Integer i = p2.first();
```

Shape Example

A

B

C

D

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

class ShapeGroup implements Shape {
 private List shapes = new ArrayList();

 ...

 public void draw(Graphics g) {
  for(Shape s : shapes) {
   s.draw(g);
}}}
```

- Which part will cause error?
- Which part should be changed?

Shape Example

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

cl
 p
  .
 public void draw(Graphics g) {
  for(Shape s : shapes) {
   s.draw(g);
}}}
```

```
ShapeGroup.java:7: incompatible types
found   : java.lang.Object
required: Shape
        for(Shape s : shapes) {
                      ^
1 error
```

D

Using Generics in Shape

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

class ShapeGroup implements Shape {
 private List<Shape> shapes = new ArrayList<Shape>();

 ...
 public void draw(Graphics g) {
  for(Shape s : shapes) {
   s.draw(g);
}}}
```

Group of Square or Circle?

Generic ShapeGroup ?

A

B

C

D

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

class ShapeGroup<T> implements Shape {
 private List<T> shapes = new ArrayList<T>();

 ...

 public void draw(Graphics g) {
  for(T s : shapes) {
   s.draw(g);
}}}
```

- Q) Now what's wrong?

Generic ShapeGroup ?

A

B

C

D

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

class ShapeGroup<T> implements Shape {
 private List<T> shapes = new ArrayList<T>();

 ...

 public void draw(Graphics g) {
  for(T s : shapes) {
   s.draw(g);
}}}
```

Are we sure T has a draw() method?

- Q) Now what's wrong?

Type Bounds

- Upper Bound on Generic Type:

```
<T extends Shape>
```

  - "T is a generic parameter which must extend Shape"

Generic ShapeGroup

```
interface Shape { void draw(Graphics g); }

class Square implements Shape { … }
class Circle implements Shape { … }

class ShapeGroup<T extends Shape> implements Shape {
 private List<T> shapes = new ArrayList<T>();
 …
 public void draw(Graphics g) {
  for(T s : shapes) {
   s.draw(g);
}}}
```

Using Generic ShapeGroup

```
public static void main(String[] args) {
 ShapeGroup<Square> sg1 = new ShapeGroup<Square>();
 ShapeGroup<String> sg2 = new ShapeGroup<String>();
 sg1.add(new Square(…));
 sg2.add("Hello World");
}
```

```
class SpecialShapeGroup<T> implements Shape {
 private ShapeGroup<T> group;
 …
}
```

- Spot the errors!!

Type Bounds

- Upper Bound on Generic Type:

```
<T extends Type>
```

  - **Type** is the name of class or interface

Type Bounds

- Upper Bound on Generic Type:

```
<T extends T1 & T2 …>
```

  - You can provide more than one!

Type Bounds

- Upper Bound on Generic Type:

```
<T2 extends List<T1>>
```

  - You can express non-trivial ones!

Generic classes
vs
Generic methods

Generic Methods

- `<type parameter> (return type) (method)`

- **Examples**
- `<T> T get(List<T> list, int index) { … }`

- `<T extends Comparable> void sort(List<T> list) { … }`

- …

Generic Methods

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

class PointCmp {
 Point min(Point p1, Point p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}

ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = min(c1,c2);
```

- Is it working?

Generic Methods

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

class PointCmp {
 Point min(Point p1, Point p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}
```

```
java: incompatible types:
generics.Point cannot be converted to generics.ColPoint
```

```
ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = min(c1,c2);
```

- Is it working?

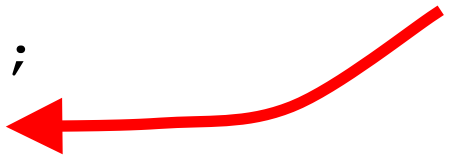
Generic Methods

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

class PointCmp {
 Point min(Point p1, Point p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}

ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = (ColPoint) min(c1,c2);
```

Needs cast on the return value!



- Can we remove casting by using generics?

Generic Methods

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

class PointCmp {
 <T> T min(T p1, T p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}

ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = min(c1,c2);
```

Generic Methods

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

class PointCmp {
 <T> T min(T p1, T p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}

ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = min(c1,c2);
```

T doesn't necessarily have x or y fields!

Generic Methods + Type Bounds

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

class PointCmp {
 <T extends Point> T min(T p1, T p2) {
  if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {
      return p1;
  } else { return p2; }
}}

ColPoint c1 = new ColPoint();
ColPoint c2 = new ColPoint();
c1 = min(c1,c2);
```

Type Erasure

- Java compiler replaces each type parameter with its first bound
- `Object` for unbounded type parameters

```
class Vec<T> {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(T e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public T get(int index) {
  if(index >= end) { throw … }
  else return (T) elems[index];
}}
```

Type Erasure

- Java compiler replaces each type parameter with its first bound
- `Object` for unbounded type parameters

```
class Vec {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(Object e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public Object get(int index) {
  if(index >= end) { throw … }
  else return elems[index];
}}
```

Type Erasure

- Java compiler replaces each type parameter with its first bound
- `Object` for unbounded type parameters

```
class Vec<T extends Comparable> {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(T e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public T get(int index) {
  if(index >= end) { throw … }
  else return (T) elems[index];
}}
```

Type Erasure

- Java compiler replaces each type parameter with its first bound
- `Object` for unbounded type parameters

```
class Vec {
 private Object[] elems = new Object[16];
 private int end = 0;
 public void add(Comparable e) {
  if(end == elems.length) { … }
  elems[end] = e; end=end+1;
 }
 public Comparable get(int index) {
  if(index >= end) { throw … }
  else return (Comparable) elems[index];
}}
```