



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



SWEN221: Software Development

14: Generics I

David J. Pearce & Nicholas Cameron & James Noble & Marco
Servetto

Engineering and Computer Science, Victoria University

What are generics?

- Introduced in Java 1.5
- 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!

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+=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?

The Generic version

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

“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

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

Shape Example

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

```
class Square implements Shape { ... }
```

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

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

```
}
```

- Q) Why doesn't this compile ?

Shape Example

A

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

```
class Square implements Shape { ... }
```

B

```
class ShapeGroup implements Shape {
```

C

```
    private List shapes = new ArrayList();
```

```
    ...
```

D

```
    public void draw(Graphics g) {
```

```
        for(Shape s : shapes) {
```

E

```
            s.draw(g);
```

```
        }
```

```
    }
```

```
}
```

- Q) Why doesn't this compile ?

Shape Example

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

```
class Square implements Shape { ... }
```

```
class
```

```
priv
```

```
...
```

```
publ
```

```
for (Shape s : shapes) {
```

```
    s.draw(g);
```

```
}
```

```
}
```

```
}
```

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

- Q) Why doesn't this compile ?

Using Generics in Shape

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

```
class Square 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);  
        }  
    }  
}
```


Generic ShapeGroup ?

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

```
class Square 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);  
        }  
    }  
}
```

Generic ShapeGroup ?

A

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

```
class Square implements Shape { ... }
```

B

```
class ShapeGroup<T> implements Shape {
```

C

```
    private List<T> shapes = new ArrayList<T>();
```

```
    ...
```

D

```
    public void draw(Graphics g) {
```

```
        for(T s : shapes) {
```

E

```
            s.draw(g);
```

```
        }
```

```
    }
```

```
}
```

- Q) Now what's wrong?

Generic ShapeGroup ?

A

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

```
class Square implements Shape { ... }
```

B

```
class ShapeGroup<T> implements Shape {
```

C

```
    private List<T> shapes = new ArrayList<T>();
```

```
    ...
```

D

```
    public void draw(Graphics g) {
```

E

```
        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

A

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

```
class Square implements Shape { ... }
```

B

```
class ShapeGroup<T extends Shape> implements Shape {
```

C

```
    private List<T> shapes = new ArrayList<T>();
```

```
    ...
```

D

```
    public void draw(Graphics g) {
```

E

```
        for(T s : shapes) {
```

```
            s.draw(g);
```

```
        }
```

```
    }
```

```
}
```

F: it is fine

Using Generic ShapeGroup

A:ok
B:error

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

- Spot the errors!!

Using Generic ShapeGroup

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

A:ok
B:error

- Spot the errors!!

Using Generic ShapeGroup

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

A:ok
B:error

- Spot the errors!!

Exact use:

`<T extends Type>`

- *Type* have to be the name of class or interface

Exact use:

`<T extends T1 & T2 ...>`

- You can provide more than one!

Exact use:

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

- You can express non trivial ones!

Generic classes
v.s.
generic methods

Generic Methods

- How to write `min()` method for subclasses of `Point`?
- How to write an auxiliary class with a method that, taking two `Point` instances, returns the one nearest the origin `(0,0)`

Generic Methods

- How to write min() method for subclasses of Point?
 - Should be possible since subclasses all have x and y fields

A

```
class Point{ int x;int y; }
```

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

B

```
class Aux1{
```

```
    Point min(Point p1, Point p2) {
```

```
        if(p1.x<p2.x || (p1.x==p2.x && p1.y<p2.y)){return p1;}
```

```
        else {return p2;}
```

```
    }
```

```
    void foo(){
```

```
        ColPoint c1 = new ColPoint();
```

```
        ColPoint c2 = new ColPoint();
```

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

```
    }}
```

D

E

F: it is fine

Generic Methods

- How to write min() method for subclasses of Point?
 - Should be possible since subclasses all have x and y fields

```
class Point{ int x;int y; }
class ColPoint extends Point{ int colour; }
class Aux1{
    Point min(Point p1, Point p2) {
        if(p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y)) {return p1;}
        else {return p2;}
    }
    void foo(){
        ColPoint c1 = new ColPoint();
        ColPoint c2 = new ColPoint();
        c1 = (ColPoint) min(c1,c2);
    }
}
```

**Needs cast on the
return value!**

Generic Methods

- How to write min() method for subclasses of Point?
 - Should be possible since subclasses all have x and y fields

```
class Point{ int x;int y; }
class ColPoint extends Point{ int colour; }
class Aux1{
    <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;}
    }
    void foo(){
        ColPoint c1 = new ColPoint();
        ColPoint c2 = new ColPoint();
        c1 = min(c1,c2);
    }
}
```


Generic Methods

- How to write min() method for subclasses of Point?
 - Should be possible since subclasses all have x and y fields

```
class Point{ int x;int y; }
class ColPoint extends Point{ int colour; }
class Aux1{
    <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;}
    }
    void foo(){
        ColPoint c1 = new ColPoint();
        ColPoint c2 = new ColPoint();
        c1 = min(c1,c2);
    }
}
```

Generic parameter is inferred

Generic Methods

- How to write min() method for subclasses of Point?
 - Should be possible since subclasses all have x and y fields

```
class Point{ int x;int y; }
class ColPoint extends Point{ int colour; }
class Aux1{
    <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;}
    }
    void foo(){
        ColPoint c1 = new ColPoint();
        ColPoint c2 = new ColPoint();
        c1 = this.<ColPoint>min(c1,c2);
    }
}
```

Generic parameter is inferred

Type Erasure!

- Java Virtual Machine doesn't know about Generics
- They are enforced only at compile time
- Can't do **instanceof** on a generic, or **new instance**
- Can break the rules with casting ...

```
int sum(ArrayList<Integer> is) {  
    int r=0;  
    for(Integer i : is) { r = r + i; }  
    return r;  
}  
  
void foo(){  
    ArrayList v = new ArrayList();    // RAW TYPE  
    v.add(new String("Crash Me!"));  
    int r = sum((ArrayList<Integer>) v);  
}
```

Finally ...

- To find more info on Generics:
 - See Sun's Java 1.5 Generics Tutorial
 - SWEN221 homepage under "Reading"
- Next time ...
 - More generics
 - Read the Tutorial!