CSC 7322: Object Oriented Development

J Paul Gibson, A207

paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7322/

Generics (in Java)

.../~gibson/Teaching/CSC7322/L10-Generics.pdf
1 Generics - Some History


David R. Musser, Alexander A. Stepanov: Generic Programming. ISSAC 1988

Charles W Kreuger, Software Reuse, ACM Computing Surveys, 1992

1 Generics - Some Java History


May 1999 - Sun proposes to Add Generics to Java, based on GJ. The activity (named JSR 14) is headed by Gilad Bracha

JSR-000014 Adding Generics to the Java™ Programming Language (Close of Public Review: 01 August 2001) http://jcp.org/aboutJava/communityprocess-review/jsr014/index.html


Recently: push for simplifying/eliminating wildcards!!!
1 A Generic Swap: in different languages

```c++
template <class T>
void swap( T& a, T& b)
{
    T tmp = a;
    a = b;
    b = tmp;
}
```

```csharp
static void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

```fsharp
let swap (a,b) = (b,a)
```

**QUESTION:** do you recognise any of these?
1 A Generic Swap: in different languages

```plaintext
generic
type Swap_Type is private; -- Generic parameter
procedure Generic_Swap(Left : in out Swap_Type; Right : in out Swap_Type);

procedure Generic_Swap(Left : in out Swap_Type; Right : in out Swap_Type) is
  Temp : Swap_Type := Left;
begnin
  Left := Right;
  Right := Temp;
nend Generic_Swap;
```

```plaintext
class Pair<T> {
  T first;
  T second;
  public static <T> void swap(Pair<T> p) {
    T temp = p.first;
    p.first = p.second;
    p.second = temp;
  }
}
```

**QUESTION**: do you recognise any of these?
1 Why are generics useful

Re-usable patterns (like higher order functions):

\[
\begin{align*}
\text{foldl } (+)\ 0\ [1..5] & = \ 15 \\
\text{foldl } (\text{append})\ ""\ ["a", \ "b", \ "c"] & = \ "\text{abc}" \\
\text{filter } (\text{odd})\ [1,3,5,2,4] & = \ [1,3,5] \\
\text{filter } (\text{animal})\ [\text{cow, dog, cake}] & = \ [\text{cow, dog}] \\
\text{map } (\text{double})\ [1,3,5,2,4] & = \ [2,6,10,4,8] \\
\text{map } (\text{capitalize})\ ["\text{aBc}, \ "\text{BBc}] & = \ ["\text{ABC}, \ "\text{BBC}] \\
\end{align*}
\]

**QUESTION**: what are the types of these 3 functions?
1 Why are generics useful

Re-usable data structures, eg binary tree of things:

With generic algorithms/functions, eg depth
1 Why are generics useful

Re-usable classes, eg (ordered) list of things:

• Combines generic data and generic functions in a **generic class**

• **Unconstrained genericity** – no restriction on type/class of generic parameter

• **Constrained genericity** – the generic parameter must be a type/class which is a subtype/subclass of a specified class

**NOTE**: Genericity is usually extended to allow multiple generic parameters (but then they may/may not be mutually constrained)
1 Why are generics useful: a classic Java example

```java
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

```java
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

**QUESTION:** Which code do you prefer, and why?

**NOTE:** The 2nd example uses the Java `List` collections class.
Why are generics useful: Java List example, continued:

```java
public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}
```

The declaration of the *formal type parameters* of the interface `List`

You might *imagine* that an `IntegerList` defined as `List<Integer>` stands for a version of `List` where `E` has been uniformly replaced by `Integer`:

```java
public interface IntegerList {
    void add(Integer x);
    Iterator<Integer> iterator();
}
```

This intuition may be useful, but it may also be misleading. (This is closer to the type of macro expansion in the C++ STL)
Java generics implemented by erasure

Generics are implemented by the Java compiler as a front-end conversion called erasure. You can (almost) think of it as a source-to-source translation (syntactic sugar), whereby the generic version of code is converted to the non-generic version.

As a result, the type safety and integrity of the Java virtual machine are never at risk, even in the presence of unchecked warnings.

Basically, erasure gets rid of (or erases) all generic type information. All the type information between angle brackets is thrown out, so, for example, a parameterized type like List<String> is converted into List. All remaining uses of type variables are replaced by the upper bound of the type variable (usually Object). And, whenever the resulting code isn’t type-correct, a cast to the appropriate type is inserted.
How To Implement Generics – many choices (see referenced papers)

While generics look like the C++ templates, it is important to note that they are not (implemented) the same.

Java generics simply provide compile-time type safety and eliminate the need for casts.

Generics use a technique known as type erasure as described above, and the compiler keeps track of the generics internally, and all instances use the same class file at compile/run time.

A C++ template on the other hand is just a fancy macro processor; whenever a template class is instantiated with a new class, the entire code for the class is reproduced and recompiled for the new class.
Some Java “Details” : all instances of a generic class have the same run-time class

What does the following code fragment print?

```java
List <String> l1 = new ArrayList<String>();
List <Integer> l2 = new ArrayList<Integer>();
System.out.println(l1.getClass() == l2.getClass());
```

It prints `true`, because all instances of a generic class have the same run-time class, regardless of their actual type parameters.

As consequence, the static variables and methods of a class are also shared among all the instances. That is why it is illegal to refer to the type parameters of a type declaration in a static method or initializer, or in the declaration or initializer of a static variable.
QUESTION: What does the following code output?

class Animal{}
class Dog extends Animal{}

public class InheritanceTester {

private static void message(Collection<Animal> animals) {
    System.out.println("You gave me a collection of animals.");
}

private static void message(Object object) {
    System.out.println("You gave me an object.");
}

public static void main(String[] args) {
    List<Dog> animals = new ArrayList<Dog>();
    message(animals);
}
}
Generics and Subtyping

In general, if Foo is a subtype (subclass or subinterface) of Bar, and G is some generic type declaration, it is not the case that G<Foo> is a subtype of G<Bar>.

All OO languages handle the integration of genericity and subclassing differently

This is probably the hardest thing you need to learn about (Java) generics … and how it relates to the concept of wildcards
Typically, a drawing will contain a number of shapes.

Assuming that the shapes are stored in a list, it would be convenient to have a method in Canvas that draws them all:

```java
public void drawAll(List<Shape> shapes) {
    for (Shape s : shapes) { s.draw(this); }
}
```

Now, the type rules (as we saw on previous slide) say that drawAll() can only be called on lists of exactly Shape: it cannot, for instance, be called on a List<Circle>.

That is unfortunate, since all the method does is read shapes from the list, so it could just as well be called on a List<Circle>…

**Java wildcards were introduced to overcome this problem.**
Wildcards – drawing shapes in a canvas

What we really want is for the method to accept a list of any kind of shape:

```java
public void drawAll(List<? extends Shape> shapes) {
  ...
}
```

There is a small but very important difference here: we have replaced the type `List<Shape>` with `List<? extends Shape>`.

Now `drawAll()` will accept lists of any subclass of `Shape` (or `Shape` itself), so we can now call it on a `List<Circle>` if we want.

`List<? extends Shape>` is an example of a bounded wildcard.

We say that `Shape` is the upper bound of the wildcard.
Java Wildcards

There are three types of wildcards in Java:

1. "? extends Type": Denotes a family of subtypes of type Type. This is the most useful wildcard

2. "? super Type": Denotes a family of supertypes of type Type.

3. "?": Denotes the set of all types or any

Question: can you think of a use of the second wildcard type?
**Problem**: Implement a Pair Of *Things* in Java

You are to code the class `GenericPair`, such that it passes the tests written in `JUnit_GenericPairTest` (which can be downloaded from the module web site).
**Problem:** Implement a Pair Of *Things* in Java

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>(package private) <code>GenericPair&lt;java.lang.Character&gt;</code></td>
</tr>
<tr>
<td>A pair of characters</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;java.lang.Character&gt;</code></td>
</tr>
<tr>
<td>A copy of the pair of Characters poc</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;java.lang.Integer&gt;</code></td>
</tr>
<tr>
<td>A pair of integers</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;java.lang.Integer&gt;</code></td>
</tr>
<tr>
<td>A copy of the pair of Integers poi</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;GenericPair&lt;?&gt;&gt;</code></td>
</tr>
<tr>
<td>A generic pair of pairs</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;GenericPair&lt;?&gt;&gt;</code></td>
</tr>
<tr>
<td>A pair of a pair of Characters</td>
</tr>
<tr>
<td>(package private) <code>GenericPair&lt;GenericPair&lt;?&gt;&gt;</code></td>
</tr>
<tr>
<td>A pair of a pair of Integers</td>
</tr>
</tbody>
</table>

**setUp**

Initialise the test variables

- poi and poi_copy as pair of Integers (0,0)
- poc and poc_copy as a pair of Characters ('a', 'b')
- popoi as (((1,2), (3,4))
- popoc as (((a,b), (c,d))
- pop as (((1,2), (c,d)))

---

2013: J Paul Gibson  
TSP: *Software Engineering*  
CSC7322/Generics.20
**Problem**: Implement a Pair Of *Things* in Java

The tests:

- **testToString**
  Tests method `GenericPair.toString()`

- **testSwap_static**
  Tests method `GenericPair.swap(GenericPair)`

- **testSwap**
  Tests method `GenericPair.swap()`

- **testCopyConstructor**
  Tests method `GenericPair.GenericPair(GenericPair)`

- **testEquals**
  Tests method `GenericPair.equals(java.lang.Object)`
**Problem:** Implement a Pair Of *Things* in Java

**TO DO:** Write the `GenericPair` so that the tests are successful

---

![JUnit Test Results]

- **Runs:** 5/5
- **Errors:** 0
- **Failures:** 0

---

2013: J Paul Gibson

TSP: Software Engineering

CSC7322/Generics.22
Problem: Implement a Pair Of Things in Java

TO DO: Write the GenericPair so that the tests are successful

You should consider the test code to specify the requirements.

For example, you can deduce that you need constructors:

- `templates.GenericPair.GenericPair(T first, T second)`
  - Explicit constructor
  - Parameters:
    - `first` is the initial value of the first element
    - `second` is the initial value of the second element

- `templates.GenericPair.GenericPair(GenericPair<T> pair)`
  - Shallow copy constructor, where first and second values are copied by reference
  - Parameters:
    - `pair` is the pair to be copied
**Problem:** Implement a Pair Of *Things* in Java (**using generics**) 

**TO DO:** Write the `GenericPair` so that the tests are successful

For example, you can also deduce that you need 2 swap methods:

```java
void templates.GenericPair.swap(GenericPair<T> p)
```

**Parameters:**

- `<T>` the type of pair elements to be swapped
- `p` the pair to be swapped

```java
void templates.GenericPair.swap()
```

Swap the first and second values of the pair

**QUESTION:** What other methods do you need?
**Problem:** Implement a Pair Of *Things* in Java (without generics)

Let us consider how we could do this *without* the generic templates

One such approach is to use the base class `Object`

```java
public interface PairSpecification {
    public void swap();
    public Object getFirst();
    public Object getSecond();
    public void setFirst(Object o);
    public void setSecond(Object o);
    public String toString();
    public boolean equals(Object obj);
}
```
Problem: Implement a Pair Of Integers in Java (3 designs)

PairSpecification

PairAbstraction

Pair Of Things

Pair Of Integers

GenericPair <Integer>

PairOfIntegersAbstraction

PairOfIntegers1
User defined generics

PairOfIntegers2
No generics

PairOfIntegers3
Library generics

ArrayList <Integer>

Integer

2013: J Paul Gibson

TSP: Software Engineering

CSC7322/Generics.26
**Problem:** Implement a Pair Of *Things* in Java

```java
public abstract class PairAbstraction implements PairSpecification{
    public abstract Object getFirst();
    public abstract Object getSecond();
    public abstract void setFirst(Object o);
    public abstract void setSecond(Object o);
}
```
**Problem:** Implement a Pair Of *Things* in Java

```java
public void swap() {
    Object temp = getFirst();
    setFirst(getSecond());
    setSecond(temp);
}
```

```java
public String toString (){
    return "(" +getFirst()+"," +getSecond()+")";
}
```
Problem: Implement a Pair Of *Things* in Java

```java
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (!(obj instanceof PairSpecification))
        return false;
    PairAbstraction other = (PairAbstraction) obj;

    return (other.getFirst() == getFirst() &&
            other.getSecond() == getSecond());
}
```
Problem: Implement a Pair Of *Integers* in Java

```java
public abstract class PairOfIntegersAbstraction extends PairAbstraction implements PairSpecification{

    public static final int DEFAULT_FIRST = 0;
    public static final int DEFAULT_SECOND = 0;

    public abstract void setFirst(Object o);
    public abstract void setSecond(Object o);

    public abstract Integer getFirst();
    public abstract Integer getSecond();
}
```
Problem: Implement a Pair Of Integers in Java

```java
public PairOfIntegersAbstraction()
    { 
    setFirst(DEFAULT_FIRST);
    setSecond(DEFAULT_SECOND);
    }

public PairOfIntegersAbstraction(PairOfIntegersAbstraction poi)
    { 
    setFirst(poi.getFirst());
    setSecond(poi.getSecond());
    }

public PairOfIntegersAbstraction(Integer first, Integer second)
    { 
    setFirst(first);
    setSecond(second);
    }
```
Problem: Implement a Pair Of *Integers* in Java (2. without generics)

```java
public class PairOfIntegers2 extends PairOfIntegersAbstraction
    implements HasInvariant, PairSpecification {

    Integer first;
    Integer second;

    public PairOfIntegers2() { super(); }

    public PairOfIntegers2(PairOfIntegers2 poi) { super(poi); }

    public PairOfIntegers2(Integer first, Integer second) {
        super(first, second);
    }

    public Integer getFirst() { return first; }

    public Integer getSecond() { return second; }
```
**Problem**: Implement a Pair Of *Integers* in Java (2. without generics)

```java
public void setFirst(Object o) throws IllegalArgumentException {
    if (!(o instanceof Integer)) throw (new IllegalArgumentException("Can't assign a non Integer"));
    first = (Integer) o;
}

public void setSecond(Object o) throws IllegalArgumentException {
    if (!(o instanceof Integer)) throw (new IllegalArgumentException("Can't assign a non Integer"));
    second = (Integer) o;
}
} // endclass PairOfIntegers2
```
Problem: Implement a Pair Of *Integers* in Java **(2. without generics)**

TO DO: Run the Unit tests and check that the implementation is correct.

QUESTION: Do you understand how the Unit tests work, what they are testing and why they are structured as they are?
**Problem:** Implement a *Pair Of Integers* (1. using user defined generics)

We can also choose to implement the PairOfIntegers using the *generic pair* as follows:

```java
public class PairOfIntegers1 extends PairOfIntegersAbstraction
implements PairSpecification {

    GenericPair <Integer> pairIntegers;

    /* TO DO – complete the code here in order that the unit tests in JUnit_PairOfIntegers1Test pass.

        The Unit test code can be downloaded from the web site
    */

}```
Problem: Implement a Pair Of Integers (1. using user defined generics)

TO DO: Implement and test a class PairOfIntegers1
**Problem:** Implement a *Pair Of Integers* (3. using library generics)

We can also choose to implement the PairOfIntegers using the generic libraries as follows:

```java
class PairOfIntegers3 extends PairOfIntegersAbstraction implements PairSpecification{
    ArrayList<Integer> pair;

    /* TO DO – complete the code here in order that the unit tests in JUnit_PairOfIntegers3Test pass.
     * The Unit test code can be downloaded from the web site
    }
```
Problem: Implement a Pair Of Integers (1. using user defined generics)

TO DO: Implement and test a class PairOfIntegers3

![JUnit Test Results]

- testEquals (0,000 s)
- testDefaultConstructor (0,000 s)
- testNonDefaultConstructor (0,000 s)
- testGetFirst (0,000 s)
- testGetSecond (0,000 s)
- testSetFirst (0,000 s)
- testSetSecond (0,000 s)
- testCopyConstructor (0,000 s)
- testSwap (0,000 s)
- testToString (0,000 s)