CSC7203 : Advanced Object Oriented Development

J Paul Gibson, A207

paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7203/

Generics (in Java)

…/~gibson/Teaching/CSC7203/CSC7203-AdvanceOO-L4-generics.pdf
1 Generics - Some History


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


1 Generics - Some Java History


Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. *Making the future safe for the past: adding genericity to the Java programming language*. SIGPLAN Not. 33, 10 (October 1998),

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 Why are generics useful

Re-usable patterns (like higher order functions):

\[
\begin{align*}
\text{foldl } (+) \ 0 \ [1..5] &= 15 \\
\text{foldl } (\text{append}) \ "\ " \ ["a", "b", "c"] &= " abc"
\end{align*}
\]

\[
\begin{align*}
\text{filter } (\text{odd}) \ [1,3,5,2,4] &= [1,3,5] \\
\text{filter } (\text{animal}) \ [\text{cow, dog, cake}] &= [\text{cow, dog}]
\end{align*}
\]

\[
\begin{align*}
\text{map } (\text{double}) \ [1,3,5,2,4] &= [2,6,10,4,8] \\
\text{map } (\text{capitalize}) \ ["aBc", "BBc"] &= ["ABC", "BBC"]
\end{align*}
\]

\textbf{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<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

```java
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) 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.
Generics and Subtyping

QUESTION: What does the following code output?

class Animal{}
class Dog extends Animal{}

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);
    }
}
Dog is subclass of Animal, List is subclass of Collection

Is a *List of Dogs* a subclass of a *Collection of 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.

**TEST: What are contravariance and covariance??**
Example: drawing shapes in a canvas

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 website). It is a good idea to put this generic class in a package reserved for generic behaviour - eg a `templates` package. I have provided JUnit tests for this class.
Problem: Implement a Pair Of *Things* in Java

### Field Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenericPair&lt;Character&gt;</td>
<td>A pair of characters</td>
</tr>
<tr>
<td>GenericPair&lt;Character&gt;</td>
<td>A copy of the pair of Characters</td>
</tr>
<tr>
<td>GenericPair&lt;Integer&gt;</td>
<td>A pair of integers</td>
</tr>
<tr>
<td>GenericPair&lt;Character&gt;</td>
<td>A copy of the pair of Integers</td>
</tr>
<tr>
<td>GenericPair&lt;GenericPair&lt;?&gt;&gt;</td>
<td>A generic pair of pairs</td>
</tr>
<tr>
<td>GenericPair&lt;GenericPair&lt;Character&gt;&gt;</td>
<td>A pair of a pair of Characters</td>
</tr>
<tr>
<td>GenericPair&lt;GenericPair&lt;Integer&gt;&gt;</td>
<td>A pair of a pair of Integers</td>
</tr>
</tbody>
</table>

The test variables

```java
void tests.JUnit_GenericPairTest.setUp()
    @Before
    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')
    - popi as ((1,2), (3,4))
    - popoc as (('a', 'b'), ('c', 'd'))
    - pop as ((1,2), ('c', 'd'))
```

The variable initialisation: setup
**Problem:** Implement a Pair Of *Things* in Java

The tests:

```java
oldTestToString
Tests method GenericPair.toString()

oldTestSwap_static
Tests method GenericPair.swap(GenericPair)

oldTestSwap
Tests method GenericPair.swap()

oldTestCopyConstructor
Tests method GenericPair.GenericPair(GenericPair)

oldTestEquals
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
**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?
Dominoes Revisited

Would it be a good idea to implement a domino as a pair of integers, re-using our generic pair behaviour?

TO DO: Implement a pair of integers, and its Unit tests, as an instantiation of our generic pair.
A generic randomly iterable class

Iterate over elements in colours of rainbow enumeration:
- colour = Red
- colour = Orange
- colour = Yellow
- colour = Green
- colour = Blue
- colour = Indigo
- colour = Violet

Iterate randomly over elements in string_data:
- three
- one
- four
- two
- five

Iterate randomly over elements in int_data:
- 2
- 3
- 1
- 4
- 5