Generic classes:

Construction, specification, verification, and static analysis

Baudouin Le Charlier
University of Namur
Belgium

Venise
march 2000
Plan of the lectures

1. Prerequisites about OO and Java

2. Construction of generic classes

3. Specification and verification of generic classes

4. Static analysis of generic classes
   (Automatic verification)
Generic classes :

Some basic notions about object oriented programming and Java
(Lecture 1)

Baudouin Le Charlier
University of Namur
Belgium

Venise
March 2000
Plan

- About these lectures
- Reusability in OO
- Inheritance in Java
- Class types in Java
- Polymorphism in Java
- Accessibility in Java
- Abstract methods and abstract classes
About these lectures

• Non topics
  – World wide web programming with Java
  – Graphical interfaces in Java
  – System programming in Java
  – Concurrent programming in Java
  – Data base access in Java
  – ...

• Topics
  – Reusability of software in Java
  – Generic classes: simulating genericity by inheritance
  – Software correctness and verification
Reusability in Object Oriented programming

• There is a lot of redundancy in “classical” programs. The same algorithm is re-coded many times with very little changes.

• Common examples: sorting and searching algorithms, classical data structures such as linked lists and binary trees.

• The main objective of object oriented programming is to reuse existing code as much as possible.

• Consequence: the coding of algorithms must be as general (generic, parametrized) as possible.
Inheritance in Java

- A class is defined by extending a previously defined class.

- The new class reuses the members (i.e., fields, methods) of the previously existing class.

- The new class may customize (i.e., modify, extend, specialize, ...) the behaviour of the old one.

- The members of the old class can be redefined (i.e., hidden, overridden) in the new one.
Class types in Java

- A *class type* is the name of a class.

- The class hierarchy defines an ordering on the types (specialization).

- A variable of type $T$ may contain a reference to an instance of any class $T'$ that is more specialized than $T$. You may write, for instance:

  ```java
  T x = new T'();
  T y = x;
  T' z = new T'();
  T w = z;
  ```

- Hence, a very general type is similar to a *parametric* type.
• Parametric polymorphism

You can write a method \( m(T_1 x_1, \ldots, T_n x_n) \) with very general parameter types \( T_1, \ldots, T_n \). The method can be applied to actual values of any types \( T'_1, \ldots, T'_n \) that are more specialized (or equal to) \( T_1, \ldots, T_n \).

• Dynamic dispatch

Every time you extend a class, you can redefine any method of it to specialize, extend, refine, ... its behaviour. The suitable version of the method will be determined at run type depending on the actual type of the object. For instance,

```java
class A{
    int m() {
        return 1;
    }
}

class B extends A{
    int m() {
        return 2;
    }
}
...
A x; ... x = new B(); ... x.m();
```

The last statement returns the value 2.
To ensure a clean separation between the specification (of the interface) of a class and its implementation, access to the members of a class can be restricted. There are four accessibility levels:

- **private**
  Accessible to code of the class only. Pure implementation notion.

- **package** (Default)
  Accessible to the package of the class. A service available in the package of the class only.

- **protected**
  Accessible in the package of the class but also to other classes that extends this (public) class provided that this member is needed to complete the implementation.

- **public**
  Accessible everywhere. A public service provided by the class.

A member cannot be redefined with a lower accessibility level.
Abstract methods and abstract classes

- **Abstract classes** are classes whose implementation is incomplete. One cannot create instances of an abstract class. Abstract classes (normally) contain at least one abstract method.

- **Abstract methods** are methods whose implementation is not provided. Only the types of the result and parameters are provided (to allow type-checking). The actual implementation will be fixed later in the actual (concrete) classes extending the abstract class.

Abstract methods can be used by normal methods. However those methods will only be applied to instances of concrete classes. Thus the abstract methods will be replaced by their implementation in the concrete class thanks to dynamic dispatch.
Construction of generic classes:

Intuitive presentation

(Lecture 2)

Baudouin Le Charlier
University of Namur
Belgium

Venise
March 2000
Generic classes

- Intuitively, a generic class is a class that is parameterized on one (maybe several) type(s) $T$ associated with a number of operations $op_1, \ldots, op_n$.

- A generic class can be instantiated to an actual class by specifying an actual type $T$ and actual operations.

- Generic classes are not provided in Java.

- Generic classes can be simulated in Java thanks to other mechanisms (i.e., inheritance, abstract methods, dynamic dispatch).

- In the second lecture, we explain how this can be done.