More about Inheritance: Method Overriding, Dispatch, and Polymorphism

T. M. Murali

Mar 17, 2004
Review

- Inheritance allows the definition of classes as extensions of other classes.
  - avoids code duplication
  - allows code reuse
  - simplifies the code
  - simplifies maintenance and extending

- A variable of type X can hold an object of type Y if and only if Y is a subtype of X.

- Substitution: a subtype can be used wherever an instance of a supertype is expected.

- All objects in Java inherit from the Object class.

- Java collections are polymorphic, can store instances of Objects.
Main Concepts to be Covered

- Static and dynamic type
- Method overriding
- Dynamic method lookup
- Method polymorphism
- Protected access
BlueJ Example: Printing in DoBE

- The `print()` method in `Molecule` only prints information in the `Molecule` class.

- Inheritance is a one-way street:
  - A subclass inherits the superclass fields.
  - The superclass does not know which new fields are defined in the subclass.
First Attempt at Solving the Problem

- Move `print()` to where it has access to the information it needs.
  - Each subclass (Gene, Protein) has its own version of `print()`.

- Two problems:
  - But Molecule’s fields are private.
  - Database cannot find a `print()` method in Molecule.
Static Type and Dynamic Type

- A more complex type hierarchy requires further concepts to describe it.
  - static type
  - dynamic type
  - method dispatch/lookup
Static and Dynamic Type

▶ The declared type of a variable is its static type.
▶ The type of the object a variable refers to is its dynamic type.
▶ The compiler checks for static type violations.
▶ Molecule mol = (Molecule) iter.next();
  mol.print(); Compile-time error.
The Solution: Method Overriding

- Superclass and subclass define methods with the same signature.
- Each has access to the fields of its class.
- Superclass method satisfies static type check.
- Subclass method is called at runtime: it overrides/redefines the superclass version.
- Why is the superclass version not invoked?
- Need to understand method dispatch/lookup algorithm in Java.
Method Lookup without Inheritance

```java
Gene gene = new Gene();
gene.print();
```

T. M. Murali: Method Overriding, Dispatch, and Polymorphism
Method Lookup with Inheritance, without Overriding

Gene gene = new Gene();
gene.print();
Method Lookup with Overriding

Molecule

print

Gene

print

instance of

Molecule mol;
mol = new Gene();
mol.print();

:Gene

T. M. Murali: Method Overriding, Dispatch, and Polymorphism
Method Lookup Algorithm

- Access the variable.
- Find the object stored in the variable.
- Find the class of the object (using dynamic type here).
- Find a matching method in the class.
- If there is no match, search the superclass.
- Repeat until there is a match or the class hierarchy is exhausted.
Super Call in Methods

- Overriding methods take precedence.
- How can we call (hidden) overridden methods?
- Use `super.method(...)`
Method Polymorphism

- A polymorphic variable can store objects of different types.
- Method calls are polymorphic too.
  - The actual method called depends on the dynamic type of the object.
Methods in the Object Class

- Methods in the `Object` class are inherited by all classes.
  - A class can override any of these.
- A common example is `toString()` method.
  - `public String toString()`
  - Returns a `String` representation of the object.
  - Allows us to omit an explicit `print()` method from a class.
  - `System.out.println(mol.toString());`
  - `System.out.println(mol);`
Protected Access

- Private access in the superclass may be too restrictive for a subclass.
- Protected access allows access to fields/methods within a class and all its subclasses.
- Protected access is more restricted than public access.
- Should fields be private or protected?
  - Recommend that they still be private.
  - Define protected accessors and mutators.