More about Inheritance: Abstract Classes and Interfaces

T. M. Murali

Mar 22, 2004
Review

- Method overriding, dispatch, and polymorphism.
- The declared type of a variable is its static type.
  - Compilers check static types.
- The type of the object held in a variable is its dynamic type.
  - Dynamic types are used at runtime.
- A subclass can override a superclass method.
- Method lookup starts with the dynamic type.
- Protected access supports inheritance.
Further Abstraction Techniques

- Abstract classes
- Interfaces
- Multiple inheritance
BlueJ Example: CellSimulator

- The CellSimulator class
- Three key components:
  - Setup in the constructor.
  - The populate method.
  - the simulateOneStep() method.

- The simulateOneStep() method.
  - Iterates over the population.
  - If the Molecule is a Gene, calls the express() method.
  - If the Molecule is a Protein, calls the fold() method.
  - The CellSimulator is tightly coupled to specific subclasses of Molecule.
Make the Molecule Superclass Abstract

- Rename methods to support information hiding:
  - `express()` and `fold()` become `act()`.
- Simulator is now significantly decoupled.
- Revised (decoupled) iteration is much simpler.
The act Method of Molecule

- Static type checking requires an act() method in Molecule.
- There is no obvious shared implementation of how a Molecule should act.
- Define act() as an abstract method:
  - abstract public void act();
Abstract Classes and Methods

- Abstract methods have the keyword abstract in the signature.
- Abstract methods have no body.
- Abstract methods make the class abstract.
- Abstract classes cannot be instantiated.
- Concrete subclasses complete the implementation.
Multiple Inheritance

- Can a class inherit directly from multiple ancestors?
- Each language has its own rules for multiple inheritance.
  - How to resolve competing definitions?
- Java forbids multiple inheritance for classes.
- Java permits multiple inheritance for interfaces.
Interfaces are Types

▶ Implementing classes do not inherit code.
▶ Implementing classes are subtypes of the interface type.
▶ Therefore, polymorphism is available with interfaces as well as classes.
Interfaces are Specifications

- Interfaces separate functionality from implementation.
  - Interfaces specify parameter and return types.
- A client that uses an interface can choose from alternative implementations.
Examples of Interfaces

- Collections interface in java.util package.
- List interface: ArrayList and LinkedList are implementations.
- Map interface: HashMap and TreeMap are implementations.
The java.io package supports input and output.

- Readers and writers deal with textual input.
  - Based on the char type.

- Streams deal with binary data.
  - Based on the byte type.
Use the FileWriter class.

Open a file: FileWriter writer = new FileWriter("name of file");

Write to the file: writer.write(<text>);

Close the file: writer.close();
Text Input

- Use the FileReader class.
- Augment with BufferedReader for line-based input.
- Open a file: BufferedReader reader = new BufferedReader(new FileReader(<file name>));
- Read from the file: String line = reader.readLine();
- Close the file: reader.close();
Review
Inheritance can provide shared implementation.
Concrete and abstract classes.
Inheritance provides shared type information.
Classes and interfaces.
Review
Abstract methods allow static type checking without requiring implementation.
Abstract classes function as incomplete superclasses.
No instances.
Abstract classes support polymorphism.
Interfaces
Interfaces provide specification without implementation.
Interfaces are fully abstract.
Interfaces support polymorphism.
Java interfaces support multiple inheritance.