Data Structures

Subodh Kumar

Dept of Computer Sc. & Engg.
IIT Delhi

More instructions

- Late policy
  - 75 hours of slack
  - Beyond this, no late submissions accepted
- Please use your IITD email
  - Entry number will be taken from address
- For Absence message
  - Mail to: col106mail@iitd.ac.in
  - Subject format must be Absent:dd/mm/yyyy
  - e.g., Absent:08/01/2015

Objects

- Program = collection of cooperating objects, with specified behavior
- Objects have properties and methods
- Send message to objects to perform a task
- Objects can contain other objects
  - Objects can send message to other objects
  - Or, create other objects
- Objects can be derived from other objects
- Object referenced by “handles”
  - Generated at creation

Object Oriented Programming

- Objects "abstract" *how-tos*
  - Interface of invocation clearly defined
- Each object knows "how to" do a few things
  - Sometimes by "taking help" from other objects
  - Does that thing only when someone asks it to
  - Becomes "inactive" after that thing is done
- Programming =
  - Decide what classes are needed
  - What objects should be created to begin with
  - And sent a message

Object Oriented Programming

- ~~Object~~Classes "abstract" *how-tos*
  - Interface of invocation clearly defined
- Each object knows "how to" do a few things
  - Sometimes by "taking help" from other objects
  - Does that thing only when someone asks it to
  - Becomes "inactive" after that thing is done
- Programming =
  - Decide what classes are needed
  - What objects should be created to begin with
  - And sent a message

Basics of Running

- **Compile:** `javac filename1.java`
  - **Output is** `filename1.class`
  - **Not machine code, but byte-code**
    - **Platform independence, more security**
  - **Executed by Java Virtual Machine**
    - **Really interpreted**
- **Run java interpreter:** `java filename1`
  - **Only the main class needs to be called**
  - **Other classes must be in the 'search path'**

Hello World!

```
import java.util.*;
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Should be in `HelloWorld.java` ?
- Executing `java HelloWorld` arg1 arg2
  - amounts to `HelloWorld.main` (args)
- `main` is public so JVM may call it
  - static, because it is independent of any instance

Hello World!

```
import java.util.*;

class Data {
    int x;
    public void print() {
        System.out.println("x = " + x);
    }
}

class HelloWorld {
  public static void main(String[] args) {
     Data a = new Data();
     a.print();
   }
}
```

Hello World!

```
import java.util.*;

class Data<T> {
    T x;
    public void print() {
        System.out.println("x = " + x);
    }
}

class HelloWorld {
  public static void main(String[] args) {
    Data<Integer> a = new Data<Integer>();
    a.print();
  }
}
```

Generic

```
List v = new ArrayList();
v.add("test");
Integer i = (Integer) v.get(0);  // Run time error
```

```
List<String> v = new ArrayList<String>();
v.add("test");
Integer i = v.get(0);  // Type error at compilation
```

Object-oriented Concepts

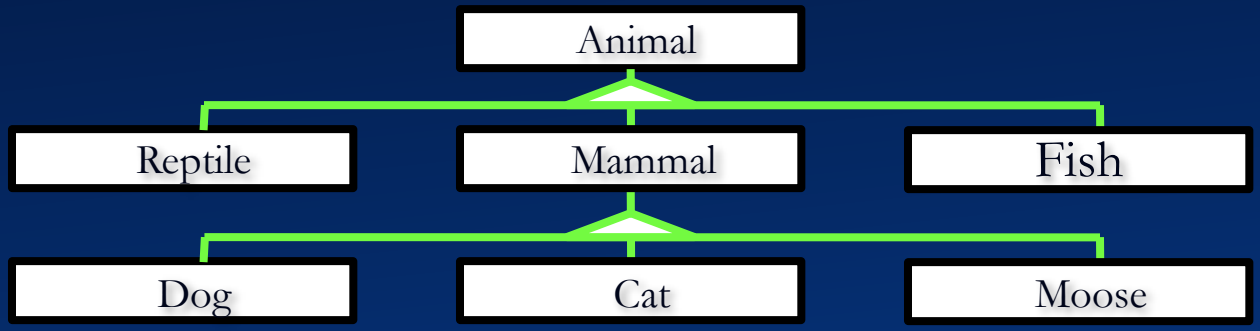
- Class and Object
  - Visibility: default, protected, private, public
  - Constructor
- Subclass
  - Inherit: “Is A” and extend, Override
- Polymorphism
  - Name overloading, Overriding
  - Use derived object in place of super-class object
- Abstract Class, Interface

Subclasses

- Vehicles
  - Moves
  - Carries passengers
  - Hold luggage
- Cars
  - Have one engine
  - Have doors
  - Has one driver
  - sportscar:
    - sleek shape, 2 doors, moves really fast, holds 2 passengers etc.
  - Van:
    - high top, 5 doors, moderate speed, holds 7 passengers, etc.

Class Hierarchy

- **Subclass models "is a" relationships**
  object of subclass "is an" object of superclass: it has all methods and attributes of the superclass
- **Inheritance Hierarchy of classes**



`Mammal` "*is an*" `Animal: Animal animal = someMammal;`

`Cat` "*is a*" `Mammal`

=> a `Cat` "*is an*" `Animal`,

`Reptile, Mammal` and `Fish` "*inherit from*" `Animal`

`Dog, Cat,` and `Moose` "*inherit from*" `Mammal`

Hierarchy Design

- superclass too general to define all behaviour
- superclass legislates an abstract behaviour and delegates implementation to subclass
- superclass specifies behaviour, subclasses inherits the behavior
- superclass specifies behaviour, subclasses can choose to override behaviour
  - It knows about super-class's behaviour

Inheritance

- A subclass specializes its superclass
  - adds new methods, overrides existing ones, and implements "abstract" methods
- A superclass factors out capabilities common to its subclasses
- A subclass inherits all capabilities of its superclass
  - `Cars` can move => `SportsCars` can move
- subclasses:
  - inherit `public` methods/variables
  - inherit `private` methods/variables
    - but cannot directly access them
    - Can access `protected` methods/variables

Abstract Methods

- Superclass delegates behaviour
  - e.g., all Vehicle instances must be able to move
    - But all Vehicle may move in its own way
- Declaration specifies method's signature
  - name, parameters and return type
  - vehicle.move() is legal with Vehicle vehicle; is declared
- Cannot instantiate an abstract class
  - A class containing an abstract method must itself be declared abstract

```
public abstract class Vehicle{
  // define to provide moving behaviour
  abstract public void move();
}
```

```
// Now this will not compile.
Vehicle myCar = new Vehicle();
```

JAVA Declaration

- To declare a subclass of `Vehicle`,

```
public class Car extends Vehicle
```

- Constructor for `Car`:

```
public Car() {
        super();
        engine = new Engine();
}
```

(In JAVA all classes automatically are subclasses of in-built `Object` class)

superclass Constructor

- Use `super` to call superclass constructor

```
public SportsCar() {
        super();
        // rest of constructor
}
```

  - Any call to the superclass constructor must be the first line of the subclass constructor
  - then, initialize any subclass instance variables
- Java's default super constructor call is implicit if no explicit call made

Overriding Methods

- Subclasses can override, i.e., redefine inherited methods:

```
public class SportsCar extends Car {
  // declarations and constructor elided
  public void startEngine() {
    // code to start engine with the
    // gusto of a SportsCar!  Vroom!
  }
}
```

- Definition of `startEngine()` replaces the one in the superclass
  - this method must be declared identically (in name and parameter list!)

Method Resolution

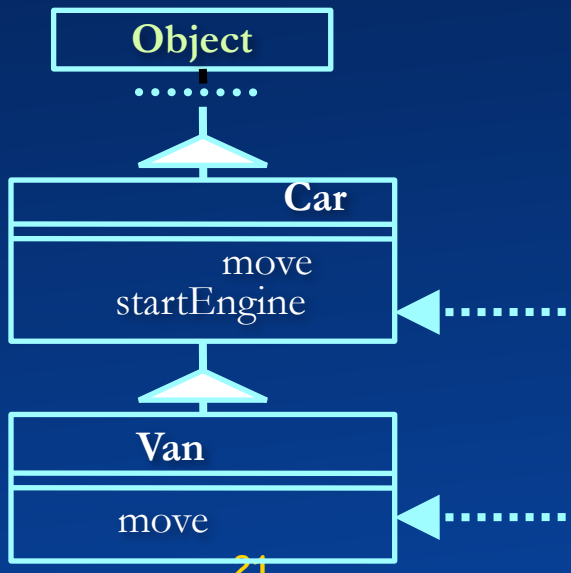
- We can use `Van` in the same way we would use `Car`
  - because it "is a" `Car`
  - `Car mysteryCar = new Van();`
- How does Java determine which method to call when `move()` message is sent to an instance?
  - `mysteryCar.move();`
  - Depends on what actual object `mysteryCar` refers to: `Van` in this case

Method Resolution Rule

- If the instance's class defines the method
  - Call it
- If not, "walk up the class inheritance tree" from class to its superclass until it either
  - finds the method: calls that inherited method
  - doesn't find the method: issues error

`startEngine()` definition found. Call it

No definition of `startEngine()` found in `Van`, looking to `Car`



```
class Aclass {
    void a()
    {
        System.out.println("aclass:a");
    }

    void b()
    {
        System.out.println("aclass:b");
        a();
    }
}

class Bclass extends Aclass {
    void a()
    {
        System.out.println("bclass");
    }
}
```

What prints? Why?

```
public class atest {
    public static void main(String argv[]) {
        Aclass avar = new Bclass();
        avar.a();
        avar.b();
```

OOP Quiz 1

- **True or False?**

1. **Subclasses inherit all methods and attributes of their superclasses** A:privates not visible
2. **Polymorphism means the programmer can create multiple instances of the same class** No
3. **Polymorphism allows different objects of a class to respond to the same message differently** super or sub can
4. **Polymorphism means that depending on the values of parameters, a method will return different values** No