The Class Concept

Object-Oriented Programming

236703

Winter 2015-6
Object Properties – Reminder

- **OOP ROCKS**

  - ID
  - Protocol → Behavior
  - Structure → State

---

**Object Properties**

- **ID**
- **Protocol → Behavior**
- **Structure → State**

**Diagram Elements**

- Car
- Steering wheel
- Speedometer
A class represents a set of objects that share a common structure and a common behavior.
Class Properties

• Dynamic Aspect (operations)
  – Protocol
    • C++: Member function declaration
  – Behavior
    • C++: Member function definition

• Static Aspect (data)
  – Structure: Data member declarations in C++.

• State and identity are usually not part of the class abstraction (why?)
  – Can a class define the state of a field?
  – Can a class define the identity of its instances?
class Stack {
    enum { N = 100 };  
    int buff[N];
    int size;
public:
    void (*push)(int element);
    int (*pop)(void);
};

• Common Parts:
  – Structure, Protocol

• Specified per Instance:
  – State: values of data members
  – Behavior: “values” of function members

• Does separating protocol and behavior make sense?
The Different Parts of a Class

• **Forge**: creation requirements
  – Signatures of accessible constructors

• **Mill**: creation process
  – Constructor bodies

• **Protocol**: communication patterns
  – Signatures of accessible fields and methods

• **Behavior**: communication effect
  – Method bodies

• **Structure**: underlying means to achieve behavior
  – Memory layout and contents of an object

Class Properties:
1. Protocol
2. Behavior
3. Structure
public class Rational {
    protected int a, b;
    public Rational(int x, int y) {
        if (y == 0)
            throw new IllegalArgumentException("y == 0");
        a = x; b = y;
    }
    public Rational() {
        this(1, 1);
    }
    public void print() {
        System.out.println(a + "/" + b);
    }
    public void add(int n) { a += n * b; }
}

In what ways are constructors different from regular methods?
public class Rational {
    protected int a, b;
    public Rational(int x, int y) {
        if (y == 0)
            throw new IllegalArgumentException("y == 0");
        a = x; b = y;
    }
    public Rational() {
        this(1, 1);
    }
    public void print() {
        System.out.println(a + "/" + b);
    }
    public void add(int n) { a += n * b; }
}
public class Rational {
    protected int a, b;
    public Rational(int x, int y) {
        if (y == 0)
            throw new IllegalArgumentException("y == 0");
        a = x; b = y;
    }
    public Rational() {
        this(1, 1);
    }
    public void print() {
        System.out.println(a + "/" + b);
    }
    public void add(int n) { a += n * b; }
}
public class Rational {
    protected int a, b;
    public Rational(int x, int y) {
        if (y == 0)
            throw new IllegalArgumentException("y == 0");
        a = x; b = y;
    }
    public Rational() {
        this(1, 1);
    }
    public void print() {
        System.out.println(a + "/" + b);
    }
    public void add(int n) { a += n * b; }
}
public class Rational {
    protected int a, b;
    public Rational(int x, int y) {
        if (y == 0)
            throw new IllegalArgumentException("y == 0");
        a = x; b = y;
    }
    public Rational() {
        this(1, 1);
    }
    public void print() {
        System.out.println(a + "/" + b);
    }
    public void add(int n) { a += n * b; }
}
The Roles Of A Class

1. A recipe for creating new objects
   – Used as part of an expression
   – In Java: `new Rational(..)`

2. A definition of a protocol
   – A contract between the class and its clients
   – Used by statically typed languages to detect contract violations prior to run time

3. An object that describes other objects
   – Run-time representation of a source-code entity
   – Used, e.g., for garbage collection

Sometimes, classes don’t have all three
Case Study – C++ Abstract Class

class Shape {
    virtual void draw() = 0;
};

• Forge, Mill
  – An abstract class may contain constructors. Used by who?

• Protocol

• Behavior
  – An abstract class may contain non abstract methods
  – Even abstract methods may have bodies!

• Structure
  – An abstract class may contain fields
Case Study – Java Interface

```java
interface Printable {
    void print();
}
```

- Forge
- Mill
- Protocol
- Behavior
- Structure

- An interface is Java’s alternatives to multiple inheritance
- Only public methods, no fields*
- Defines a type, but cannot be instantiated
Case Study – Java Anonymous Class

```java
Printable p = new Printable() {
    public void print() {
        ...
    }
};
p.print();
```

- An anonymous class is a nameless extension/implementation of another class/interface
- Only a single instance of its type
- Can add and override methods, but extensions’ visibility is limited*

*Requires the extension’s visibility to be at least package-private or public.
Case Study – Java Anonymous Class

```java
Printable p = new Printable() {
    private int x;
    { x = 3; } // Instance initializer
    public void print() { System.out.println("Hello " + f()); }
    int f() { return x; }
    Printable init(int val) { x = val; return this; }
}.init(int_exp);
```

- **Forge**
  - A single object of this class can be created

- **Mill**
  - Using instance initializer or the `init()` idiom (*only semantically* in Mill)

- **Protocol, Behavior**
  - Externally: Only that of the base class/interface
What About Squeak Class?

Object subclass: #Point
instanceVariableNames: 'xCoord yCoord' ...

initialize
  xCoord := 0. yCoord := 0! !

x
  ^xCoord! !

x: newX
  xCoord := newX! !

• Forge, Protocol
  ▪ Not known at compile time – yes, not even the constructor!

• Mill, Behavior

• Structure
A Class As An Object

• Classes can have a run time representation
  – Needed for dynamic dispatch, garbage collection, serialization...
  – C++, Eiffel: very limited, if any
  – Java, C#, all dynamic languages: A class is an object
    • It has a protocol that may be used to inspect the class, create new objects and sometimes change the definition of the class
Reflection In Java

void assignField(Object o, String name, Object value) {
    Class c = o.getClass();
    Field f = c.getField(name);
    f.set(o, value);
}

This function:
• Takes an object, o
• Sends the getClass() message to this object
• Gets a result, c
• Sends the getField() message to the result, c
• Sets the value of the field

• (Exception handling omitted – but why might one be thrown?)
Reflection In Java

Is the result of `getClass()` an object?

• If no:
  – Then how can we send the message `getField()` to it?

• If yes:
  – Then we can apply `getClass()` again. What would the following expression yield then?

```java
o.getClass().getClass().getClass().getClass()
```
The Two Principles Of Reflection

1. Every object is an instance of a class
2. Every class has an object that represents it

So:
- We take a “starting” object, o.
- The object o was instantiated from a class
- There is an object o1 that represents the class that instantiated our starting object o.
- The object o1 was instantiated from a class
- There is an object o2 that represents the class that instantiated the object o1 that represents the class instantiated our starting object o.
- The object o2 was instantiated from a class
- There is an object o3 that represents the ...
Implementing Reflection

In Java:

\[ o2 == o3 == o4 == \ldots \]

- This means that there is an object that instantiated itself
- The class that it represents is called **Meta-class**
- This object is created differently than other objects

Programming languages may be classified by the level of meta-classes – we shall discuss this later.