Java and C# in Depth

Exercise Session – Week 3
Agenda

- Assignment I Review
- Class Initialization and Class Instance Creation
- Quizzes
- Assignment II Handout
Double Dispatching in Visitor Pattern

```java
public class ReversePolishNotationCalculator {
    /** Stack of current operands */
    private Stack<Node> stack;
    /** Operation evaluator */
    private Evaluator evaluator;
    ...
    /** Push Plus operator on top of stack. */
    public void pushPlusOperator() {
        stack.add(new PlusOperator());
    }
    /** Evaluate stack. */
    public void evaluateStack() throws Exception {
        Node n = stack.pop();
        n.accept(evaluator);
        pushOperand(evaluator.getResult());
    }
}

public class PlusOperator extends Operator {
    ...
    public void accept(Visitor visitor) throws Exception {
        visitor.visitPlusOperator(this);
    }
}

public interface Visitor {
    public void visitPlusOperator(PlusOperator o) throws Exception;
    ...
}

public class Evaluator implements Visitor {
    /** Evaluation result */
    private double result;
    /** Stack to evaluate on */
    private Stack<Node> stack;
    public Evaluator(...) {
        // Initialize stack ...
    }
    public void visitPlusOperator(PlusOperator o) throws Exception {
        double d1 = getNextOperand();
        double d2 = getNextOperand();
        result = d1 + d2;
    }
}
```

Dynamic binding
Class Instance Creation in Java

Creating a new class instance
- Storage allocation for all the fields (this + super)
- All instance variables initialized to default values
- Instance initialization (process the constr.)
  1. If starts with explicit/implicit super constr. invocation, process the super constr. (by applying 1–4 recursively to super constr.), go to 3
  2. If starts with explicit this constr. invocation, process this constructor (recursively), go to 4
  3. Execute all the instance initializers and instance variable initializers in their textual order
  4. Execute the rest of constructor body

```java
class A {
    A(String s)
    { System.out.println(s); }
}
class E {
    A a1=new A("a1");
    A a2;
}
class F extends E {
    A a3=new A("a31");
    { a3=new A("a32"); }
    A a4;
    F() {
        this(5);
    }
    F(int i)
    { a4=new A("a4"); }
    // what's the output?
    F f=new F();
}
```
Class Initialization in Java

- Initializing a class
  - When:
    - Before the first “use” of class
    - The direct superclass, but not the implemented interfaces, must be initialized before a class
    - Attributes defined in interfaces are initialized when accessed for the first time
  - How
    - Compile time constants initialized first
    - Static initializers and initializers for static fields are executed in textual order and only once
  - “Use” of class
    - Class instance creation
    - Static member reference
    - Invocation of certain reflective methods in class Class and in package java.lang.reflect

```java
// with the same class A

class B {
    static A s1 = new A("s11");
    static {
        s1 = new A("s12");
    }
}

interface D {
    A s2 = new A("s2");
}

class C extends B implements D {
    static {
        s3 = new A("s31");
    }
    static A s3 = new A("s32");
}

// what's the output?
C c = new C();

s11
s12
s31
s32
```
Creating a new class instance

- Storage allocation for all the fields (this + super)
- All instance variables initialized to default values
- Instance initialization (process the constr.)
  1. If it has an explicit/implicit constr.-initializer of form base(...)
     a. Execute the instance variable initializers in their textual order
     b. Process base constructor * (recursively)
     c. Go to 3
  2. If it has explicit constr.-initializer of form this(..)
     a. Process this constructor (recursively)
     b. Go to 3
  3. Execute the rest of constructor body

```csharp
//with the same class A
class E {
    A a1=new A("a1");
    A a2=new A("a2");
}
class F : E {
    A a3=new A("a31");
    A a4=new A("a41");
    public F():this(5) {}
    public F(int i)
    {a4=new A("a42");}
}
// what's the output?
F f=new F();
```

Class A: `A a1=new A("a1");`  
Class B: `A a2=new A("a2");`  
Class C: `A a3=new A("a31");`  
Class D: `A a4=new A("a41");`  
Class E: `A a1=new A("a1"); A a2=new A("a2");`  
Class F: `A a3=new A("a31"); A a4=new A("a41"); public F():this(5) {} public F(int i) {a4=new A("a42");}`
Class Initialization in C#

- Initializing a class
  - **Static constr.**
    - Its execution is triggered by first “use” of class
  - **Static field initializer**
    - Executed right before the static constructor, if any;
    - Otherwise, before first class member reference
    - In their *textual order* and *only once*

- “Use” of class
  - Creation of an instance of the class
  - Reference to any of the static members of the class

// with the same class A
class B {
    static A s1 = new A("s11");
    static B()
    { s1 = new A("s12"); }
}
class C : B {
    static C() {
        s3 = new A("s31");
    }
    static A s3 = new A("s32");
}
// what's the output?
C c=new C();
s32
s31
s11
s12
Comparison

- **Initialization code**
  - (Java) Multiple static or non-static block initializers, constructors
  - (C#) Static and non-static constructors

- **Instance variable initialization**
  - (Java) superclass -> subclass
  - (C#) subclass -> superclass

- **Class variable initialization**
  - (Java) superclass -> subclass
  - (C#) subclass -> superclass

- a1, a2, s1, s2 are from superclass; a3, a4, s3, s4 are from subclass
Quiz 1: Class Initialization Dependence

```java
class G {
    static int i = f();
    static int j = 7;
    static int f() {
        return j;
    }
}

class G {
    public static int i = f();
    public static int j = 7;
    static int f() {
        return j;
    }
}

public class Initialization{
    public static void main(String[] args){
        System.out.printf("G.i=%d, G.j=%d", G.i, G.j);
    }
}

public class Initialization{
    public static void main(String[] args){
        System.out.printf("G.i=%d, G.j=%d", G.i, G.j);
    }
}

G.i=0, G.j=7

Problem: With wrong order of static initializer, it is possible to observe a static field before it is initialized to the chosen value.

Recommended practice: Using static methods to initialize the class variables in proper order.
```
Quiz 2: Object Initialization and Polymorphism

```java
public class Point {
    protected final int x, y;
    private final String name;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        // 3. Invoke subclass method
        name = makeName();
    }

    protected String makeName() {
        return "["+x+", "+y+"]";
    }

    public final String toString(){
        return name;
    }
}

public class ColorPoint extends Point {
    private final String color;

    public ColorPoint(int x, int y, String color){
        // 2. Chain to Point constructor
        super(x, y);
        // 5. Initialize blank final
        // Too late!
        this.color = color;
    }

    protected String makeName() {
        return super.makeName() + ":" + color;
    }

    public static void main(String[] args) {
        // 1. Invoke subclass constructor
        System.out.println(
            new ColorPoint(4, 2, "purple");
        }
    }
}
```

[4,2]:null
Quiz 2: Object Initialization and Polymorphism (Cont.)

- Constructor calls a method overridden in its subclass
  - The method runs before the instance has been *fully* initialized

**Problem:** With calls to overridden methods in a constructor, it is possible to observe the state of an object before it is fully initialized.

- Recommended practice
  - Never call overridable methods from constructors
  - Lazy initialization vs. eager initialization

```java
public class Point {
    protected final int x, y;
    private final String name;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        // name = makeName()
    }

    protected String makeName() { ... }

    public String getName() {
        if (name == null) name = makeName;
        return name;
    }

    public final String toString() {
        return getName;
    }
}
```
Questions?
Assignment 2

- See published pdf
- Java implementation due before 11 March
- C# implementation due before 18 March