Object interactions
Lecture 5

Waterford Institute of Technology

March 26, 2014

John Fitzgerald

Abstraction and Modularization

House implementable with
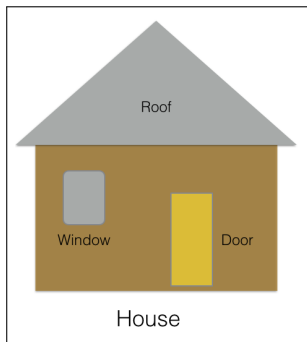primitive types only
Better use House class
Has objects of class types:

- Window

- Door

- Roof

Approach known as *abstraction*

- Abstract from details

- Reduces complexity

- Modularization facilitates
  specialization

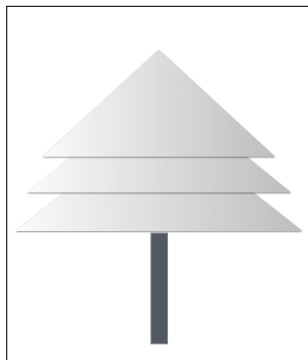- *Divide and conquer*
  recommended approach

Abstraction and Modularization

Tree object implementable with primitive types only

Better use Tree class
- *Has* objects of class types:
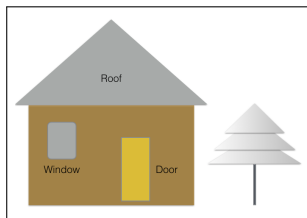 - Triangle
  - Rectangle

Simple example scaleable to much more complex problems

Abstraction and Modularization

Specialization

- Developer **Tree** class unaware of **House**

- House developed much later than Tree

- House developer not concerned with internals of Tree

- Only interested in interface to Tree

-   - How to create Tree object
    - Available behaviours, methods Tree

Creating objects

new operator followed by constructor creates object

- `Tree oakTree = new Tree();`
- `House ourHouse = new House();`
- `House yourHouse = new House(Window w, Door d);`
- `Point origin = new Point(100.34, 200.67);`

These statemtents have three components

- Variable declaration `Tree oakTree`
- Instantiation using `new Tree()`
- Initialization of object within constructor

Objects create objects

Objects may create objects

```java
public class BankAccount
{
    Customer customer;
    public BankAccount(int customerID)
    {
        this.customer = new Customer(customerID);
    }
}
```

Here a Customer object

- Instantiated in BankAccount constructor
- Initialized with actual parameter *customerID*

Mutiple Constructors

Multiple constructors permitted

```java
public class Rectangle
{
    private int width = 0;
    private int height = 0;
    private Point origin;

    public Rectangle() {
        setState();
    }
    public Rectangle(int width, int height) {
        setState(int width, int height);
    }
    private void setState(int width, int height) {
        this.width = width;
        this.height = height;
        origin = new Point(0,0);
    }
}
```

Internal method calls

Method can invoke other
method same class

Class *Rectangle* has

- Two public
  constructors
    - Default
    - Overloaded
- Private helper
  method *setState*

```java
public Rectangle()
{
    setState();
}

public Rectangle(int width, int height)
{
    setState(int width, int height);
}
```

External method calls

Method can invoke other method different class

Class *Tree* has

- Field *Triangle treeBase*
- *treeBase* object has *createTreeBase* method
- *setState* invoked within *createTreeBase*
- This an example external method call

```
public class Tree
{
    private Triangle treeBase;
    ...
    public void createTreeBase(int height, int width)
    {
        treeBase = new Triangle();
        treeBase.setState(height, width);
    }
```

DRY principle

Do **N**ot **R**epeat yourself

Assemble code into reusable units

```java
//verbose code
public class Point
{
    private int x;
    private int y;

    public Point() {
        this.x = 10;
        this.y = 10;
    }
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```java
//DRY code
public class Point
{
    private int x;
    private int y;
    public Point() {
        setState(10,10);
    }
    public Point(int x, int y) {
        setState(x, y);
    }
    private void setState(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

The **null** object
A billion $ mistake

null a Java object

- Object default initialization *null*
- Can test for *null*
- *java.Lang.NullPointerException*
    - Attempt operation on null object

```java
private Tree tree;

public void drawTree()
{
    if(tree == null)
    {
        tree = new Tree();
    }
    else
    {
        draw(tree);
    }
}
```

Error handling

Java uses **exception** event to handle runtime errors

Example of **exception** event

- Tree object declared
- Default initialization **null**
- **new** operator not invoked
- Attempt operation on *null*
  - *java.Lang.NullPointerException*

```
private Tree tree;
public void drawTree()
{
    tree.draw();
}
```

Using BlueJ Debugger

Debugger software application

- Integral to most IDEs

- Step-by-step code execution

- Can step into and out of methods and constructors

- Breakpoints insertable

- Execution halts at breakpoint

- Class, instance and local variable values then available

- Provides information on *call sequence* or *stack*
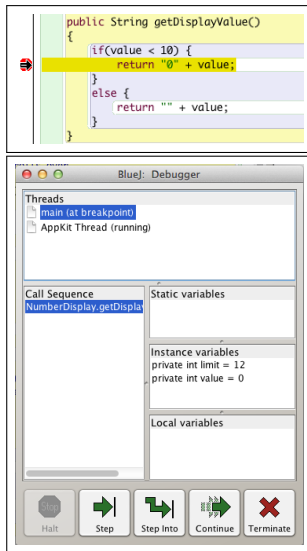
Using BlueJ Debugger

Debug into object

- Open class source code
- Place cursor where breakpoint required
- Tools Set / Clear Breakpoint
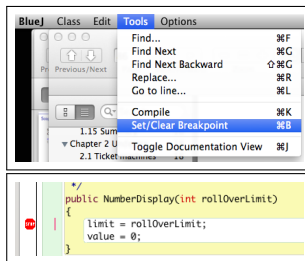
Using BlueJ Debugger
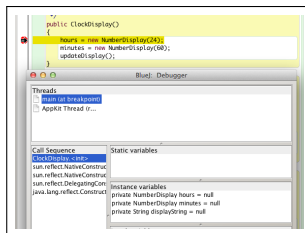

In BlueJ instantiate *new* obect

- Invoke method where breakpoint
- Execution halts at breakpoint
- Displays debug window
- Variable data visible

Using BlueJ Debugger

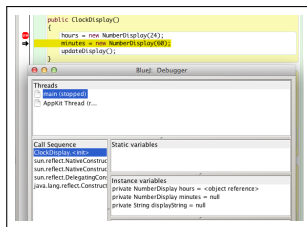ClockDisplay constructor
Before initialization

- All instance variables *null*

Using BlueJ Debugger
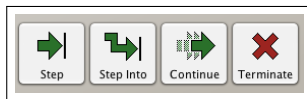
ClockDisplay constructor
During initialization

- *hours* new object reference assigned

- *minutes* still default *null*

Using BlueJ Debugger

Debugger options

- **Step:** Execute next statement
- **Step into:** Step into method
- **Continue:** Run to next breakpoint or end
- **Terminate:** Quit debugging

Referenced material

1. *null* reference: a billion dollar mistake
http://en.wikipedia.org/wiki/Tony_Hoare [Accessed
2014-03-26]