Java (Core Java)

Encapsulation

Today's Topics

- Encapsulation
  - Packages
  - Access Modifiers
  - Beans

Encapsulation

- Encapsulation simply means controlling access of class attributes and methods to only the rightful group.
- For example:
  - Even in the last seen shape example, the individual shapes should not be allowed to change noOfSides of the parent class Shape.
  - Imagine what would happen if in Square we changed the noOfSides of Shape to 3, or 10
  - You might want some attributes of the super class to only be visible to its sub classes but not anything else
  - You might just want to organize your code and keep related classes together in a “folder” or a “group”
  - What if you want to keep a class read only or write only? For instance if you want to create a class to read from a file and that file will never be updated.

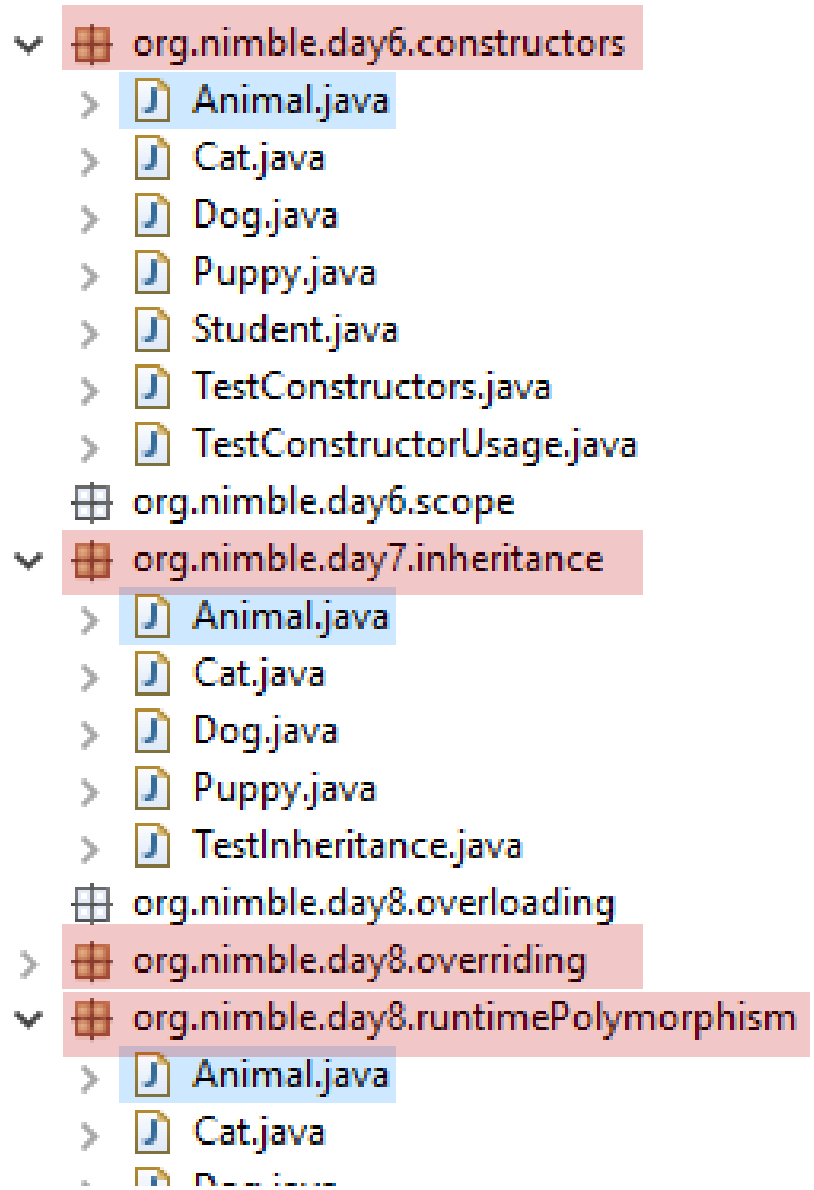
Encapsulation

- Following are the two themes in Java which incorporate the concept of encapsulation
  - Packages
  - Access Modifiers
- In Java a completely encapsulated class has all its instance variables private and is called a Bean. We will learn about it later.

Packages

- Package is a folder or a group of classes, usually related.
- Allows you to have duplicate class names (in different packages). For instance, look at classes named Animal, Cat, Dog appearing so many times in different packages. In one package you cannot have two classes with same name.
- Package can have sub-package under it

- Advantages:
  - Organized code
  - Removes name collisions
  - Provides protection (we will see later)



Packages

- To indicate that a class is part of a package, we use the package keyword at the top of the class.
- A class from a package can be referenced in another class in three ways:
  - Using import at package level
  - Importing the exact class
  - Not importing, but using the fully qualified class name

```java
package org.nimble.day7.packages; // tells which package this class belongs to

import org.nimble.common.dataObjects.Student; // imports specific class
import org.nimble.day7.inheritance.*; // imports all classes of the package

public class ReferencingClassesFromOtherPackage {

    @SuppressWarnings("unused")
    public static void main(String... args) { // This is varargs.
        Student[] students = new Student[3];
        Animal anAnimal = new Animal();

        // What if I wanted to create another Animal but belonging to another package?

        org.nimble.day6.constructors.Animal anotherAnimal = new org.nimble.day6.constructors.Animal();
    }
}
```

Question:
What will happen in above program if I also imported another package where there is a class called Animal, or Student?

Packages

- To compile a class using packages, follow below steps:

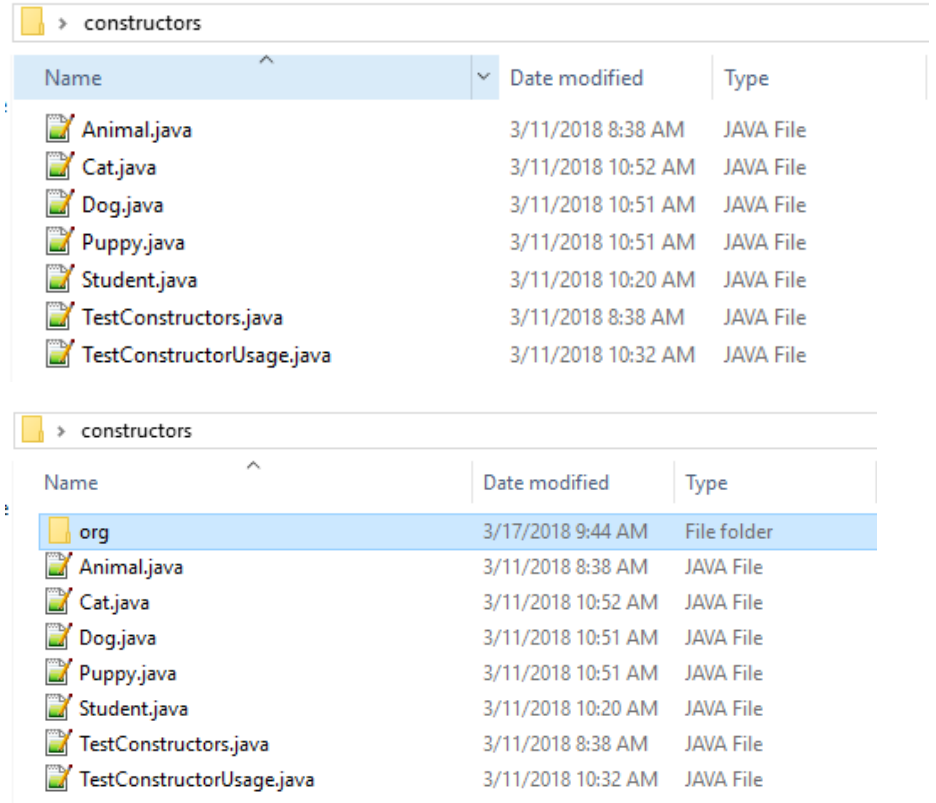
- To compile

  Go to the directory

  To compile, `javac -d . FileName.java`

```
C:\Users\TANU\Desktop\constructors>javac -d . Student.java
```

- To execute
  - Go to the directory
  - To execute, `java FullyQualifiedClassName`




```
C:\Users\TANU\Desktop\constructors>java org.nimble.day6.constructors.Student
Start
Creating a student
Created st1.  Name : null Gender :   Roll No : 0 Year of birth : 0 Age : 0 Curr Class : 3
Variable st2 created
Variable st3 created
```

Access Modifiers

- There are four types of access modifiers
  - private
  - default
  - protected
  - public

Access modifier	Within class	Within package	Outside package by subclass only	Outside package
private	Y	N	N	N
default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

Access Modifiers

```
package org.nimble.day7.accessmodifiers;

public class Shape {
    protected int noOfSides; // Why protected? Why not private?
    protected int lengthOfOneSide; // Why protected? Why not private?

    public void perimeter() { // Why public? Why not protected?
        System.out.println("Perimeter is : " + this.noOfSides * this.lengthOfOneSide);
    }

    public void area() {
        System.out.println("Not enough info..");
    }

    // Parameterized Constructor
    Shape (int noOfSides, int lengthOfOneSide){
        this.noOfSides = noOfSides;
        this.lengthOfOneSide = lengthOfOneSide;
    }

    // Default constructor
    Shape(){
    }
}
```

Access Modifiers

```
package org.nimble.day7.accessmodifiers;

public class Rectangle extends Shape {

    private int lengthOfShorterSide; // Why private? Why not anything else?
    private int lengthOfLongerSide;

    // Overriding the area of Shape with more specific implementation
    public void area() {
        System.out.println("Area is : " + lengthOfLongerSide * lengthOfShorterSide);
    }

    // Overriding the area of Shape with more specific implementation
    public void perimeter() {
        System.out.println("Perimeter is : " + 2 * (lengthOfLongerSide + lengthOfShorterSide));
    }
```

Access Modifiers

```
package org.nimble.day7.accessmodifiers;

public class Square extends Shape{

   String color; // What does this mean? What kind of access this has?

    // Why public?
    public void area() {
        System.out.println("Area is : " + Math.pow(lengthOfOneSide, 2));
    }

    // Default constructor
    Square (){
        this.noOfSides = 4;
    }

    // Overridden constructor
    Square (int lengthOfEachSide){
        this.noOfSides = 4;
        this.lengthOfOneSide = lengthOfEachSide;
    }
}
```

Complete Encapsulation - Beans

- Bean is a class where all the attributes are declared as private
- There are getters and setters to read and update the values of the attributes.
- A bean with only getters is a read only class
- A bean with only setters is a write only class

```
public class StandardBean {
    private String className;
    private Student[] students;
    private Teacher teacher;

    public String getClassName() {
        return className;
    }
    public void setClassName(String className) {
        this.className = className;
    }
    public Student[] getStudents() {
        return students;
    }
    public void setStudents(Student[] students) {
        this.students = students;
    }
    public Teacher getTeacher() {
        return teacher;
    }
    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }
}
```

Complete Encapsulation - Beans

- Write Only class – keep only the setters
- Read only class – keep only the getters
- Typically in beans, attribute manipulation is done through other public, private and protected methods.

```
public class StandardBean {
    private String className;
    private Student[] students;
    private Teacher teacher;

    public String getClassName() {
        return className;
    }
    public void setClassName(String className) {
        this.className = className;
    }
    public Student[] getStudents() {
        return students;
    }
    public void setStudents(Student[] students) {
        this.students = students;
    }
    public Teacher getTeacher() {
        return teacher;
    }
    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }
}
```