CS2

Object Oriented Programming in Java

Handout : Week2

prepared by N S kumar

Preface

Dear friends,
Trust you read, understood and enjoyed the lecture notes of week 1.

I am giving you the next installment of the lecture notes.

There could be some differences in the content delivered in a particular week. We plan to include these topics in the following weeks in case some of these have been missed in that particular week.

You may send your feedback to the email id listed below. Your inputs will be helpful in making the lecture notes better.

N S Kumar
Visiting Professor
Dept. Of Computer Science and Engineering
PES Institutions.
Email : ask.kumaradhara@gmail.com
Date : 18th Jan 2015

Drive Into Objects and Classes:

In the first week, we learnt the following:

- why programming is required?
- What are the different paradigms of programming?
- How high level languages evolved?
- What are the major characteristics of Object Oriented Programming?
- What are the salient features of Java?

More importantly, we learnt the concept of interface and implementation. We said that an interface should never change and that the implementation could change.

Let us start with a program to find the area of a triangle given the base as 10 and the height as 20.

```
// Area of a triangle of base 10 and height 20
public class AreaOfTriangle1
{
    public static void main(String[] args)
    {
        System.out.println(1 / 2 * 10 * 20);
    }
}
```

We use the formula(we call it an expression – nothing to do with the one on your face now as you read this – hopefully smiling?) to find the area of triangle and give that to println to print(we call that an argument). We expect to get 100. When we compile the program and run, we get the Indian contribution to Mathematics – ZERO!

Why did we get zero? We are using operators / for division and * for multiplication. There are plenty operators in Java. An operator in a programming language has the following:

precedence:

This is somewhat similar to BODMAS rule that we learn in arithmetic, but more complicated as there are lots of operators in these programming languages. The order of evaluation of operators depends on the precedence. Operators with higher precedence will be evaluated first.

Association:

What if more than one operator in an expression has the same precedence? In that case, we evaluate left to right or right to left depending on the operator. This concept is called association.

Rank or arity:

The number of operands required for an operator is called arity. Some operators are unary(one operand) – some are binary(two operands) and one of them is ternary(three operands).

In this example, operators of division and multiplication have the same level of precedence and are evaluated left to right – left associative.

An expression has a value – of a particular type. This is decided by the language.
Division of integers in many languages results in an integer. So 1 / 2 becomes 0 and now you can figure out why we get zero.

To get a fractional result in division, we may use floating values(real in math) in division.

```
// Area of a triangle of base 10 and height 20
public class AreaOfTriangle2
{
    public static void main(String[] args)
    {
//      System.out.println(1 / 2 * 10 * 20); // WRONG
        System.out.println(1.0 / 2.0 * 10 * 20);
    }
}
```

Pay attention to the precedence, the association of operators and the type of result of expression evaluation.

You may download the Java Tutorial from Oracle. This is a very good tutorial to learn Java.
http://docs.oracle.com/javase/tutorial/

The above program finds the area of a triangle of base 10 and height 20. What if I want to find area of triangle with different bases and heights. You may remember that in the school days, we learnt arithmetic and then algebra. In algebra, we learnt the concept of variables. Variables could take different values. How do we use the same idea in programming?

```
// base and height are variables to which we can values from the keyboard
import java.util.*;
public class AreaOfTriangle3
{
	public static void main(String[] args)
	{
		int b;
		int h;
		System.out.print("enter base and height : ");
		Scanner in = new Scanner(System.in);
		b = in.nextInt();
		h = in.nextInt();
		System.out.println(1.0 / 2.0 * b * h);
		in.close();
	}
}
```

In this example, the user can specify the base and the height while running the program. There are quite a few new items in this program.

To take input from the keyboard, we have to refer to it in the program. It is called System.in. System.out refers to output and System.in refers to input. Reading from the keyboard is complicated as we may decide to input a number or a string or special characters. To handle this, we use a type(class) called Scanner that Java provides. We make a variable of that type, bind it to the keyboard and then we will ask it to get us the next integer.

```
Scanner in = new Scanner(System.in);
```

We will very shortly discuss about the new operator.
This Scanner is available in something called java.util – utilities java provides.
Hence the line:

```
import java.util.*;
```

```
int b;
int h;
```

What would these statements do? We are saying that we want variable called b and h and that their type is int. A variable has the following characteristics.

- Name
- location in memory
- type which indicates
  - range of values – therefore the size
  - set of operators
- value in that location

This list is not complete. We will discuss more of them later.
In this example, b is a variable of type int. This statement causes allocation of memory of 4 bytes. Java supports 8 types – called primitive types – byte, char, short, int, long, boolean, float and double. Unlike 'C', size of each type is fixed in Java. Can you find the size of each of these types?

These variables are created within a block of a function. These are called local variables. These variables at this point would not have any value – have a state - uninitialized variable. Unless given a value to the variable, we cannot use them.

We put a value into the variable after getting it from the keyboard. We use the value of the variable in the formula or expression to compute the area.

Variable has a name, a location, a type and a value. Local variables are not initialized by default in Java. Use of uninitialized variable causes compile time error in Java.

What if I want to find the area of two triangles? Simple. Create variables – base and height for triangle 1 and again base and height for triangle 2!

```
// find areas of two triangles
import java.util.*;
public class AreaOfTriangle4
{
	public static void main(String[] args)
	{
		int b1;
		int h1;
		int b2;
		int h2;
		Scanner in = new Scanner(System.in);
		System.out.print("enter base and height : ");
		b1 = in.nextInt();
		h1 = in.nextInt();
		System.out.print("enter base and height : ");
		b2 = in.nextInt();
		h2 = in.nextInt();
		in.close();
		System.out.println("area of triangle 1 : " + 1.0 / 2.0 * h1 * b1);
		System.out.println("area of triangle 2 : " + 1.0 / 2.0 * h2 * b2);
	}
}
```

Thats a bad idea. What if I had a polygon with n sides (n > 3)? We may have to specify n sides and a few more elements to define the figure uniquely. In stead of carrying books separately, you may want to carry them in a bag. Then we can talk about the book in Rama's bag or the book in Krishna's bag. Why not create

}

Triangle as a type with length and breadth as attributes or fields? Here is the next example with user defined type.

```
// make triangle a separate type

import java.util.*;
public class AreaOfTriangle5
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		Triangle t1 = new Triangle();
		System.out.print("enter base and height : ");
		t1.b = in.nextInt();
		t1.h = in.nextInt();
		Triangle t2 = new Triangle();
		System.out.print("enter base and height : ");
		t2.b = in.nextInt();
		t2.h = in.nextInt();
		System.out.println("area of triangle 1 : " + 1.0 / 2.0 * t1.h * t1.b);
		System.out.println("area of triangle 2 : " + 1.0 / 2.0 * t2.h * t2.b);
		in.close();
	}
}


class Triangle
{
	public int h;
	public int b;
}
```

Primitive and Reference Variable:

Let us discuss a few aspects of this code. What does this mean?

```
    Triangle t1;
```

and how does this compare with

```
    int b;  ?
```

```
    int b;
```

This statement causes allocation of memory for b where an integer can be stored. If the variable is local – in a block within a method -, it will not be initialized.

```
    b = 10;
```

would put 10 into the location associated with b.

```
    Triangle t1;
```

On the other hand, the above statement creates only a single location for t1 and t1 at that point does not have allocation for height and base.

We have a key, but no site and no house!

We create the site – space for fields of Triangle – by using the operator new.

```
     t1 = new Triangle();
```

After this execution, we can refer to t1.b and t1.h.

We have a site now and but still no house.

We should put something into b and h before using it. This is the way we do – we have built our house.

```
    t1.b = in.nextInt();
    t1.h = in.nextInt();
```

We can also put something into another triangle the same way.

We can now compute the area.

Whenever we have to represent a thing or a concept, we make that a separate type or class.

Each of these classes can be in their own files. A program in Java can consist of number of files – each file having one or more classes. A file can have only one public class and in that case, the file name should match the class name.

Why should we develop programs in multiple files? This would allow development by different people, therefore would make the development faster. This also allows for reuse. The class developed can be used by others. This concept is called a library in programming parlour.

You may remember that our city name changed recently from Bangalore to Bengaluru. Your course CS2 is not called CS2 : Theme : Object Oriented Programming in Java. What is constant in this world is change. What would happen if we change the field names of the class Triangle from b to base and h to height. We want to make our field names more meaningful.

```
public class Triangle
{
//	public int b;
//	public int h;
	public  int breadth;
	public  int height;
}
```

Now the client program (the one using this Triangle) will not compile. This client program depends on the Triangle class. The Triangle class should specify clearly what will not be changed(interface) and what could change(implementation).

If the designer of the Triangle class expects that the field name and/or type could change over a period, he should not expose(make them public). He should provide functions(methods) to play with the class Triangle.

Let us have look at the changed Client and Triangle classes.

```
// filename: Triangle.jav
public class Triangle
{
        private int breadth;
        private int height;
        public void initialize(int b, int h)
        {
                this.breadth = b;
                this.height = h;
        }
        public double area()
        {
                return (1.0 / 2.0) * breadth * height;
        }
}

// filename: Client.java
import java.util.*;
public class Client
{
        public static void main(String[] args)
        {
                Scanner in = new Scanner(System.in);
                int b; int h;
                Triangle t1 = new Triangle();
```

```
		System.out.print("enter length and breadth of a triangle : ");
		b = in.nextInt();

		h = in.nextInt();
		t1.initialize(b, h);
		Triangle t2 = new Triangle();
		System.out.print("enter length and breadth of a triangle : ");
		b = in.nextInt();
		h = in.nextInt();
		t2.initialize(b, h);

		System.out.println("area of triangle 1 : " + t1.area());
		System.out.println("area of triangle 2 : " + t2.area());
		in.close();
	}
}
```

Observe that the Client does not have to know how the Triangle stores the attributes. He calls a method called initialize to put values into the attributes.
t1.initialize(b, h);

Similarly the client calls the method area to get the area. He does not have to know how this method computes the area.
t1.area()
This is a very simple example of encapsulation.

We put together related attributes and behaviour.
We hide what could change and we expose what shall not change.
We hide the implementation and expose the interface.

Before we can use any object, it has to be initialized. We here used a method called initialize to initialize the object. Almost all Object Oriented Programming

Languages provide a special function called the constructor to initialize the object.

In Java, the constructor is a method of the class. Has the same name as the class. Has no return type. Can take parameters. Is always called when an object is created. Here is an example with the constructor. We will discuss more of this next week.

```
public class Triangle
{
        private int breadth;
        private int height;
        public double area()
        {
                return (1.0 / 2.0) * breadth * height;
        }
        // Constructor
        public Triangle(int b, int h)
        {
                breadth = b;
                height = h;
        }
}
```

The client code needs to be changed to call the constructor in this fashion.

```
        //t1.initialize(b, h);

        Triangle t1 = new Triangle(b, h);
```

We will revisit these concepts at length during the lectures of next week.

Operators, Operands, Types and Expressions:

The concept of operators which comes from Math is also central to any programming language. We already said that operators have precedence, association and arity. Here are some questions to ponder.

- Can we use an operator on any type?

  Can we add two integers? Can we add two floating point numbers?

  Can we add an integer and a double?

  If we add operands of two different types, what type will be the result?

- How would the following expressions be evaluated?

  3 + 4   3.5 + 4.6   3 + 4.6   3.5 + 4

  Would the computer evaluate these the same way?

- When an operator is usable on operands of different types, we say that the operator is overloaded. Operator + can take a numeric operand or a string operand. How about these expressions as argument to println? Can we use them outside of this context?

  3 + “mooru”

  “mooru” + 3

  3 + 4 + “elu”

  “elu” + 3 + 4

- Assignment in Java is very strict about the use of operands of different types. For example, we can assign an int to a double and not the other way. You may want to experiment with different possible types to understand the Java Philosophy with respect to assignment operator.

- In 'C', we use the terms l-value and r-value expressions. L-value refers to the one on the left of assignment and r-value refers to the one on the right of assignment. In 'C', dereferencing operator returns l-value back – so can occur to the left of assignment – hope you are not seeing stars! Can we use an expression to the left of assignment?

- In 'C', the order of evaluation of operands is not defined. Value of an expression of the form a * a++ is not defined as the value of the left a can be fetched before incrementing or after incrementing. What happens in Java?

- Java also have a rich set of operators – including those for bitwise operations. What do these operators do?
  >> >>>
- We can write an expression a + b + c (assume that all are integers). Based on the rules of association, the computer will evaluate the expression left to right or right to left. In this case, it is left to right. The computer finds the result of a + b and then adds c to it. This concept is also called cascading. Can we use this expression in Java?
  3 > 2 > 1
  Can we use > operator on boolean type?
  Can we cascade == operator on boolean type?

- Will the expression given below result in runtime error – division by zero?
  int a = 0; int b = 10;

  boolean c = (a == 0) || (b / a &gt; 5);
  if so why? If not also the same question!
  Have you come across a concept called short circuit evaluation or don't care in Kmap?

Let the above puzzles keep you puzzling for some time!

We shall conclude this week notes with a repeat of the discussion of the Object Based Programming Paradigm.

abstraction

selecting essential features
ignoring non-essential features
abstraction depends on the observer

encapsulation

putting attributes and behaviour together
hide what could change; hide the implementation
expose what shall not change; expose the interface
access control

composition

attribute of the class being an object of a class
reuse mechanism.

We shall consider an example or two to illustrate these ideas.

In my childhood days, we would challenge each other with puzzles – called ogatu in Kannada. One of them was this.
“I have 3 eyes ; I am not Shiva
I have water; I am not a pot
I have hair; I am not a woman”.

I am sure you can easily guess the answer – Coconut. Describing coconut to somebody who has never seen it so difficult. How difficult it is to describe an elephant to somebody who has never seen it? Abstraction introduces a new terminology and raises the level of abstraction. Simple Idea. Try explaining your credit based grading system to your elders at home who have never heard of it.

Abstraction depends on the observer. Give your mobile phone to a small child. A very small one immediately would taste (not test – Or it is its way of testing!) it.
A little bigger one will try whether it can break it – will bang it to the ground or throw it. Some times I use my mobile phone as a paper weight. In one of the movies (a rare feat!) I have seen, an empty bottle dropped from a aeroplane causes havoc in that small human settlement (movie: Gods must be crazy).

We said earlier that abstraction depends on the observer. A fan circulates air. One possible abstraction of the fan could include operations like start the fan, stop the fan, change speeds or change direction. Some of you may want to put the fan on so that the voice of the teacher is lost in the sound the fan makes! For a fan manufacturer, the abstraction would be different. He may think in terms of blades, motors. He puts these components together to make the fan – this concept is called composition. A fan contains motor, blades, regulator and so on. For a seller of the fan, fan is just a product to be sold for money.

When we analyze the problem, we arrive at an abstraction based on what are the expected behaviours. To support these behaviour, we will include some attributes and methods in our type. The following discussion could highlight these concepts better.

Concept of the time:
The concept of the Time is an interesting example of a real world entity which we would like to model as a type. What attributes does a time have? hours? minutes? what about seconds? If we are covering a race of snails, would we require seconds? If we are timing a rocket launch, would we require something smaller than a second? Is this concept used to find the time period between two events? Is it used to indicate the current time? If we talk about the current time, should we also consider the time zone? What if the country supports day light saving? Should the clock be a 24 hour clock or a 12 hour clock? should we store the strings am or pm to indicate whether it is before noon or after noon? Or shall we have both 24 hour time as well

as 12 hours time with indication of am or pm? Can we store time all in seconds counting from the beginning of some era? What about the operations? should we support displaying? should we support addition of a time period to time? Should we support difference of times? Shall we also consider the way time is stored as per the Indian system - Ghalige and vighalige?

Assuming that we decide to store the time as hours, minutes and seconds - with the condition that we may have to change them to a single number if required later – what type should these be? should the field names be known to the user? should the user be allowed to change minutes directly? Lots of questions for you to think with no single answer - but we can most of the time make out the good ones from the bad ones!

In the morning, We have classes so early - College does not allow us to have a good sleep- especially does not allow us to have early morning dreams - I understand that "early morning dreams come true". So, I get up in the morning - spend a few seconds brushing my teeth, wash my face and have a good breakfast in front of the TV(that is a must - how can we forgo football and filmy stuff?). Then I pack my bag with some books (a few have to be left behind!) and then rush to the college. Can I make a class (not the one in the college!)- my own type - to capture all these?

```
class MorningBlue:
attributes:
        toothpaste
        toothbrush(may not be mine!)
        soap
        bread toast
        coffee
        TV
        book - (may be books)
        bag - (to put all together)
```

This is considered a bad idea. We have put together things which are not necessarily related to each other. TV can be shared with siblings (of course fight over who gets the remote control). TV does not go into my bag. If we put together attributes which are not related, we say that we have created a God class. We say that the class lacks cohesion. Lesser mortals like us may have divide this into smaller ones.

We may want a class(one for each) to represent

- those related to cleaning activities
- those related to breakfast
- those related to takeToCollege
- those related to visual activities - some of you may read the newspaper instead of watching the stupid box.

Thats it for this week. Enjoy.