Introductory Programming and Object-oriented Concepts Using Java

Unit 5

Classes and Objects

A Dice Game - Example

- To generate a random number from 1 – 6 to represent the value of a dice:

```
int value=(int)(Math.random() *6) + 1;
```

- A loop to roll the dice 10 times:

```
for ( int n=1; n<=10; n++) {
  int value=(int)(Math.random() *6)  + 1;
  System.out.println(value);
}
```


Comments on Dice Game

- Need to know how Math.random() works
- Variables declared may be together with other variables in the application.

Another approach to the Dice game

```
public static void main(String[] args)
{
   Dice d = new Dice();
   for ( int n=1; n<=10; n++)
   {
        d.roll();
        System.out.println( d.getValue() );
   }
}
```

d

Comments on the Dice game

- Models after a real dice
- New - creates a dice
- roll – simulates rolling a dice
- getValue() - gets the face value of the dice

Object Oriented Programming

- Dice is an example of a **class** (a template)
- Dice d = new Dice();
  d is an **object**, or the 'real' dice
- Dice d1 = new Dice(); creates another dice d1
- d.roll();
  roll() is a **behaviour** or action of Dice

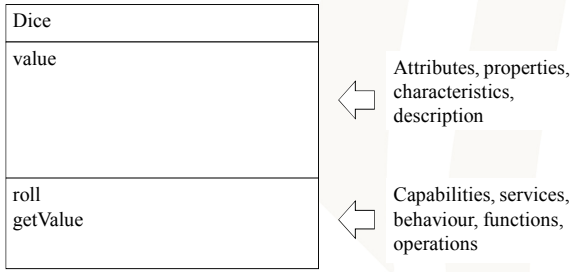
Object Oriented Programming

- Programming that models after real life situations
- Put all related variables and methods together (Abstraction)
-
- Hides all details. Expose only through method call (Encapsulation)

Object Oriented Programming

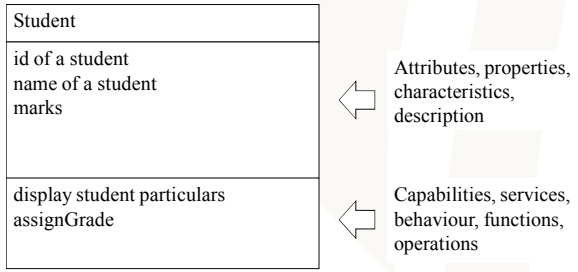
- Class
  - A structure where we define all related variables belonging to a entity.
  - All related methods that process the variables
  - Only a template, actual object not created yet
- Objects or instances are ***actual*** entities
  - Object = identity + instance variables + methods

Basic Structure of a Class



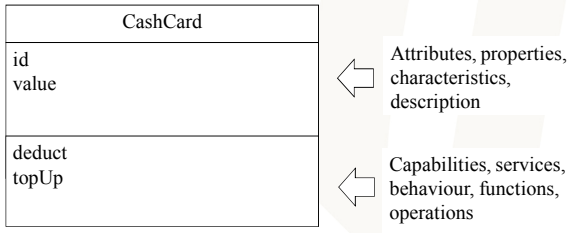
9

Basic Structure of a Class



10

Another Example



11

Writing a Class

```
public class CashCard
{
  //instance variables
  //constructor
  //accessor methods
  //methods
}
```

12

Instance Variables

```
public class CashCard
{
    private String id;
    private double value;

}
```

- Include only attributes relevant to the application

Constructor

- To initialize the values of the instance variables
- It has a method name that has the same name has the class name
- No return type

Constructor

```
public class CashCard
{
    private String id;
    private double value;

    public CashCard(String id, double amount) {
        this.id=id;
        value = amount;
    }
}
```

- same name as class name
- no return type
- what is **this**?

Creating objects

```
public class CashCardApp {
    public static void main(String[] args ) {
        CashCard c1 = new CashCard("1", 10.0);
        CashCard c2 = new CashCard("2", 20.0);

    }
}
```

Same sequence as the constructor parameters

- 2 sets of Cash card created

Multiple constructors

```
public class CashCard
{
    //instance variables

    public CashCard(String id,  double amount) {
          …
    }

    public CashCard(String id) {
          this.id = id;
          value= 10.0;                  //fix 10 dollar cash cards
    }
}
```


Accessor methods

```
public class CashCard
{
    //instance variables
    //constructor

    public String getId() {
          return id;
    }
     public void setId(String id){
          this.id = id;
    }
    public double getValue() {
          return value;
    }
}
```

- What happen to the setValue method?

Other methods - Behaviour

```
public class CashCard {
    …
    public void topUp(double amt) {
          value += amt;
    }

    public void deduct(double amt) {
      if ( value >= amt)
          value -= amt;
    }
}
```


Sending message to object

Format: object.*message*(parameters);

```
CashCard myCard = new CashCard("123", 10.0);

myCard.deduct(2.5);

myCard.topUp(10.0);

System.out.println( myCard.getValue() );
```

toString method

```
public class CashCard
{
   …

   public String toString() {
        return “Id: “ + id + “Value: ” + value);
   }
}
```

- Usually returns the attribute values as a String

Why toString method?

```
CashCard myCard = new CashCard(“123”, 10.0);

// to print details of cash card

System.out.println( myCard.toString() );

// However, with toString method, can do this:

System.out.println( myCard );
```


Array of Objects

- Similar to array of primitive types
- E.g.

```
CashCard[] cards = new CashCard[10];
cards[0] = new CashCard(“11”, 10);
…

for ( int i=0; i<cards.length; i++)
   System.out.println( cards[i].getValue() );
```

Objects not created yet!

Random Class

- Instead of having to know Math.random() method, a class is available that hides the details of how a random number is generated.
- Create a Random object:

```
Random r = new Random();
```

- To generate an integer value:

```
int n = r.nextInt(10);    // n will be 0 to 9
```

Dice class

```
import java.util.*;

public class Dice
{
   private int value;
   private Random r;

   public Dice()
   {
      r=new Random();
      value = r.nextInt(6) + 1;
   }

   public void roll()
   {
       value = r.nextInt(6) + 1;
   }

   public int getValue()
   {
      return value;
   }

}
```

Static keyword

- Keyword static can be applied to variables and methods when writing a class
- Static variables are class variables
- Static method are class methods

Static methods

- Static methods are class methods
- Do not require an object to call such methods
- Example is Math class. It is a collection of static methods. There is no constructor in this class to instantiate a Math object
- To call class methods, use
  - <class name>.<method>
  - E.g. double n = Math.sqrt(4);

Static methods

- Since static methods are called using class names, such methods CANNOT reference any instance variables.
- Instance variables can also call static methods

Static variables

- Static variables are variables defined in a class preceded by the keyword static
- There is only 1 copy of this variable during execution versus the many copies of instance variables for every object instantiated
- For example, for the CashCard class, for a top up amount of 100 dollars or more, the cash card has additional 1% in value.
- 1% applies to all cash card top ups, so it should not be an instance of every CashCard object

Static variables - Example

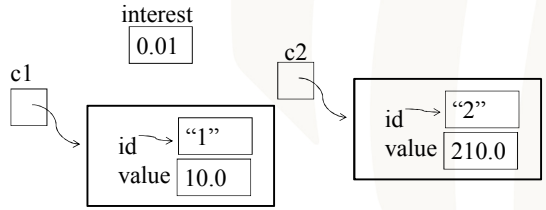
```
public class CashCard
{
        private static interest=0.01;
        private String id;
        private double value;

        public CashCard(String id, double amt) {
          this.id = id;
          if ( amt >=100)
                value = amt * (1+interest);
          else
            value = amt;
        }
}
```


Static variables – Example (cont'd)

```
public class CashCardApp {
   public static void main(String[] args ) {
        CashCard c1 = new CashCard("1", 10.0);
        CashCard c2 = new CashCard("2", 200.0);

   }
}
```



Modifiers

- Defines the scope where variables of methods can be accessed

	Within a class	Outside of a class
private	Yes	No
public	Yes	yes

Modifiers

```
public class Aclass
   private int x;
   public int y;
   …
  private void methodA(){
    //can access x, y
  }

  public void methodB(){
    //can access x, y
  }
```

```
public class MainClass
   public static void main(…){
     Aclass a = new Aclass();

     a.x = 10;          //ok?
     a.y= 10;           // ok?

     a.methodA();       //ok?
     a.methodB();       //ok?

  }
```

33