CE/CZ2002:
OBJECT ORIENTED DESIGN & PROGRAMMING

Tutorials

SCHOOL OF COMPUTER SCIENCE & ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Tutorial 1

Object Oriented Concepts & Basic Java

1. Based on your understanding, identify, with reasons, whether the following are mostly considered as Class, Object, Attribute (properties) or Behaviour ?
(note : all words are deliberately capitalized)

Student	NTU	Book	MichaelJackson	Age
Color	Work	Person	Person1	Result
Transformer	Engine	Liquid	Force	Shoot

2. Considering *School* as a *Class* and taking an example of our school SCE, identify as many classes (at least 5) as possible relating to the School class. Show the attributes and behaviours of each class. You may draw out a hierarchy of classes.

3. Convert the following Bubble sort program in *C language* code to *Java language* code by :
   a. Identify the line number of the code to be changed
   b. Replace with the Java language syntax.
   c. Suggest any improvement to be made to the program

```c
1   #include<stdio.h>
2
3   void bubble(int a[],int n)
4   {
5       int i,j,t;
6     for(i=n-2;i>=0;i--)
7         {
8         for(j=0;j<=i;j++)
9             {
10              if(a[j]>a[j+1])
11                  {
12                  t=a[j];
13                  a[j]=a[j+1];
14                  a[j+1]=t;
15              }
16          }
17      }
18   }//end function.
19
20   int main()
21    {
22       int a[100],n,i;
23
24      printf("\n\n Enter number of Integer elements to be sorted: ");
25      scanf("%d",&n);
26
27      for( i=0;i<=n-1;i++)
28      {
29          printf("\n\n Enter integer value for element no.%d : ",i+1);
30          scanf("%d",&a[i]);
```

```
31              }
32
33          bubble(a,n);
34
35          printf("\n\n Finally sorted array is: ");
36          for( i=0;i<=n-1;i++)
37                  printf("%d  ",a[i]);
38      } //end program.
```
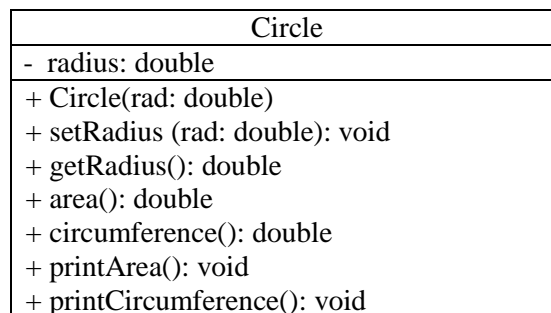
Tutorial 2

Classes & Objects

1. Write a class `Circle` that has the following instance variables and methods:

```
public class Circle
{
    private double radius;        // radius of circle
    private static final double PI = 3.14159;

    // constructor
    public Circle(double rad) {...}
    // mutator method – set radius
    public void setRadius(double rad){...}
    // accessor method – get radius
    public double getRadius(){...}
    // calculate area
    public double area(){...}
    // calculate circumference
    public double circumference() {...}
    // print area
    public void printArea(){...}
    // print circumference
    public void printCircumference(){...}
}
```
The UML class diagram for the Circle class is given below:

Circle
- radius: double
+ Circle(rad: double) + setRadius (rad: double): void + getRadius(): double + area(): double + circumference(): double + printArea(): void + printCircumference(): void

Write an application class `CircleApp` to test the `Circle` class. The class `CircleApp` should display a menu. The user can then select an option of the following: (1) create a new circle; (2) print area; (3) print circumference; and (4) quit. Implement the operations for each option.

A sample program run is given below:

```
 ----jGRASP exec: java CircleApp

==== Circle Computation =====
|1. Create a new circle     |
|2. Print Area              |
|3. Print circumference     |
|4. Quit                    |
=============================
Choose option (1-3):
1
Enter the radius to compute the area and circumference
```
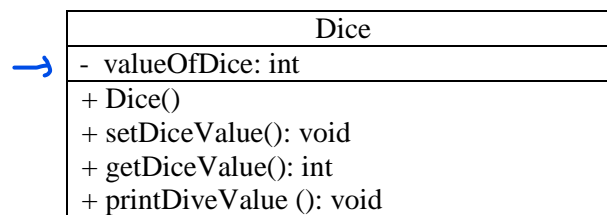
3

```
5
A new circle is created
Choose option (1-3):
2
Area of circle
Radius: 5.0
Area: 78.53975
Choose option (1-3):
3
Circumference of circle
Radius: 5.0
Circumference: 31.4159
Choose option (1-3):
4
Thank you!!
```

2. Write a class `Dice` that has the following instance variables and methods:

The UML class diagram for the Dice class is given below:

Dice
- valueOfDice: int
+ Dice() + setDiceValue(): void + getDiceValue(): int + printDiveValue (): void

Write an application class `DiceApp` to test the class `Dice`. The class `DiceApp` interacts with the user to generate the numbers randomly from rolling a pair of dices. The generated numbers from the pair of dices and the total are then displayed on the screen.

A sample program run is given below:

```
 ----jGRASP exec: java DiceApp
Press <key> to roll the first dice
1
Current Value is 3
Press <key> to roll second dice
2
Current Value is 3
Your total number is: 6
```
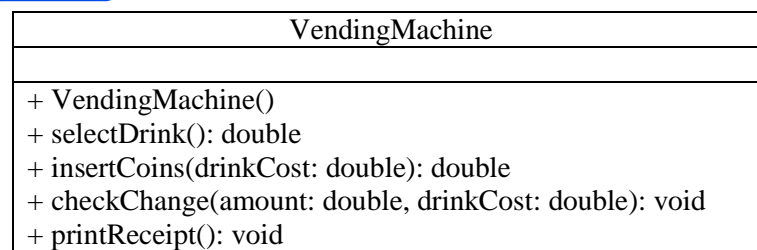
Tutorial 3

Class Methods & Inheritance

1. Design a program to implement a vending machine for buying drinks. Write a class `VendingMachine` that has the following methods:

```
public class VendingMachine
{
    // constructor
    public VendingMachine() {}

    // get the drink selection, and return the cost of the drink
    public double selectDrink() {...}
    // insert the coins and returns the amount inserted
    public double insertCoins(double drinkCost) {...}
    // check the change and print the change on screen
    public void checkChange(double amount, double drinkCost)  {...}
    // print the receipt and collect the drink
    public void printReceipt(){...}
}
```

The UML class diagram for the VendingMachine class is given below:

VendingMachine
+ VendingMachine() + selectDrink(): double + insertCoins(drinkCost: double): double + checkChange(amount: double, drinkCost: double): void + printReceipt(): void

Write an application class `VendingMachineApp` to test the class `VendingMachine`. The program allows users to select the drink to buy, and accept coins inserted by the user to pay for the drink. The program will also print the receipt for user to collect the drink.

a) Discuss the design of the VendingMachine class and how it can be improved.
b) What will be a relevant class to relate to the VendingMachine class?
c) Suggest how the application can be re-designed to involve the class in (b)?
d) [Optional] Implement your design.
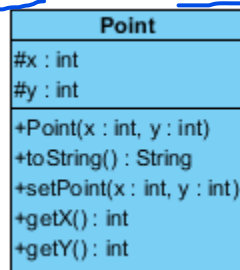
A sample program run is given below:

```
====== Vending Machine ======
|1. Buy Beer ($3.00)        |
|2. Buy Coke ($1.00)        |
|3. Buy Green Tea ($5.00)   |
|===========================
Please enter selection:
1
Please insert coins:
========== Coins Input ===========
```

```
|Enter 'Q' for ten cents input    |
|Enter 'T' for twenty cents input|
|Enter 'F' for fifty cents input  |
|Enter 'N' for a dollar input      |
=================================
Q
Coins inserted: 0.10
T
Coins inserted: 0.30
F
Coins inserted: 0.80
N
Coins inserted: 1.80
N
Coins inserted: 2.80
N
Coins inserted: 3.80
Change: $ 0.80
Please collect your drink
Thank You !!
```

2. You are given the class diagram for the Point class :

Point
#x : int
#y : int
+Point(x : int, y : int)
+toString() : String
+setPoint(x : int, y : int)
+getX() : int
+getY() : int

const π

The *toString*() method will return the *x* and *y* value in the format "[x, y]".

X. Write the code in Java.

2. Create a class Circle to inherit from the Point class. The Circle class is to have the following methods: **setRadius**, **getRadius, toString** and **area**. Reuse whatever you can from the Point class. πr^2
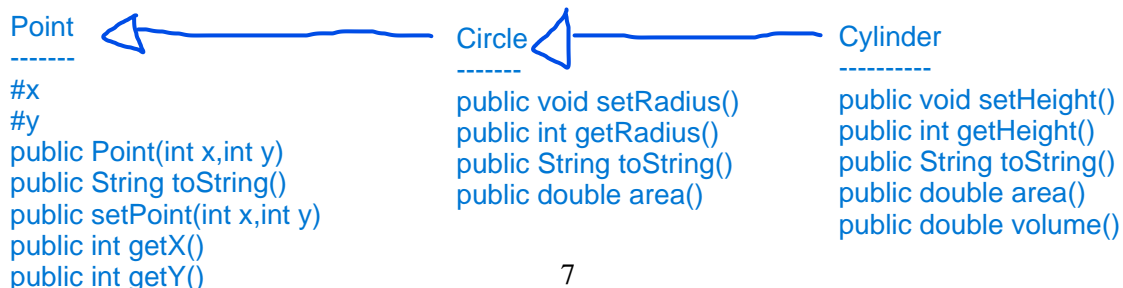
3. Create a class Cylinder to inherit from the any of the classes above. The Cylinder class is to have the following methods: **setHeight**, **getHeight**, **toString, area** and **volume**. → $\pi r^2 h$

$2\pi r^2 + 2\pi r h$

4. Draw the class hierarchy. Create and use instances of a circle and a cylinder to test classes you have created. Do you think that it was a good choice to use Point as the base class? Suggest alternatives.

Point	Circle	Cylinder
-------	-------	----------
#x	public void setRadius()	public void setHeight()
#y	public int getRadius()	public int getHeight()
public Point(int x,int y)	public String toString()	public String toString()
public String toString()	public double area()	public double area()
public setPoint(int x,int y)		public double volume()
public int getX()		
public int getY()		

7

Tutorial 4

Exception Handling (*eLearning*)

1. What output is produced by the following code? What would be the output if waitTime were 12 instead of 46?

```
int waitTime = 46;
try {
    System.out.println("Try block entered");
    if (waitTime > 30)
        throw new Exception("Time Limited Exceeded");
    System.out.println("Leaving try block");
}
catch (Exception e)
{
    System.out.println("Exception: " + e.getMessage());
}
System.out.println("After catch block");
```

2. Define an exception class called PowerFailureException. The class should have a constructor with no parameters. If an exception is thrown with this zero-argument constructor, *getMessage* should return "Power Failure!". The class should also have a constructor with a single parameter of type String. If an exception is thrown with this constructor, then *getMessage* returns the value that was used as an argument to the constructor.

3. What is the output produced by the following program? What would the output be if the argument to sampleMethod were -99 instead of 99? What would it be if the argument were 0 instead of 99?

```
public class NegativeNumberException extends Exception
{
    public NegativeNumberException()
    {
        super("Negative Number Exception!");
    }
    public NegativeNumberException(String message)
    {
        super(message);
    }
}
public class FinallyDemo
{
    public static void main(String[] args)
    {
        try {
            sampleMethod(99);
        }
        catch(Exception e)
        {
            System.out.println("Caught in main.");
```

wont print if Exceotion class with no arguments called

skip rest of try once Exception thrown

go back to catch after finally block

Q3a)
Caught in main
In finally block
Still in sampleMethod
After finally block
 Caught in main

b)
Negative Number Exception!
Caught in sampleMethod
Still in sampleMethod
In finally block
After finally block

c)
No Exception
Still in sampleMethod
In finally block
After finally block

8

```
      }
    }
    public static void sampleMethod(int n) throws Exception {
      try  {
        if (n > 0)
          throw new Exception( );
        else if (n < 0)
          throw new NegativeNumberException( );
        else
          System.out.println("No Exception.");
        System.out.println("Still in sampleMethod.");
      }
      catch(NegativeNumberException e)
      {
        System.out.println("Caught in sampleMethod.");
      }
      finally
      {
        System.out.println("In finally block.");
      }
      System.out.println("After finally block.");
    }
  }
```

4. Write a program that implements a simple calculator. The calculator keeps track of a single number (of type double) that is called *result* and that starts out as 0.0. The user is allowed to repeatedly add, subtract, multiply, or divide the *result* by a second number. The result of one of the operations becomes the new value of *result*. The calculation ends when the user enters the character 'Q' or 'q' for quit. In addition, if the user enters any operator symbol other than +, -, *, or /, the UnknownOperatorException is thrown and the user is asked to reenter that line of input. You are also required to define the class UnknownOperatorException.

   A sample dialog of using the calculator is given below:
   Calculator is on
   result = 0.0
   +
   5
   result + 5.0 = 5.0
   updated result = 5.0
   *
   2.2
   result * 2.2 = 11.0
   updated result = 11.0
   %
   10
   % is an unknown operator
   Please reenter:
   updated result = 11.0
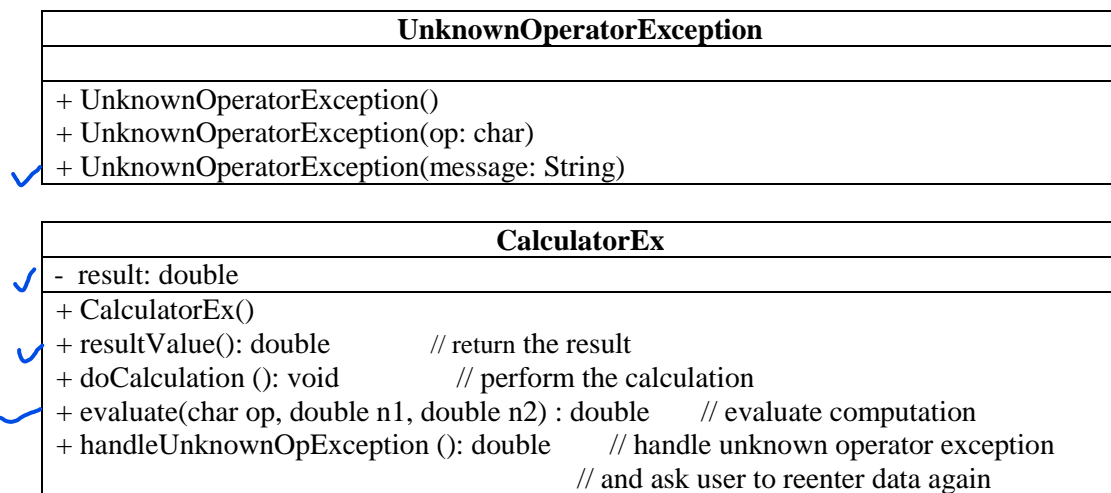   *
   0.1
   result * 0.1 = 1.1

updated result = 1.1
q
Final result = 1.1
End of Program

The UML diagram for the class UnknownOperatorException and the class ClaculatorEx is given below. Also write the static main() method in the CalculatorEx class when starting the calculator.

UnknownOperatorException
+ UnknownOperatorException()
+ UnknownOperatorException(op: char)
+ UnknownOperatorException(message: String)

CalculatorEx
- result: double
+ CalculatorEx()
+ resultValue(): double   // return the result
+ doCalculation (): void   // perform the calculation
+ evaluate(char op, double n1, double n2) : double   // evaluate computation
+ handleUnknownOpException (): double   // handle unknown operator exception
// and ask user to reenter data again

Tutorial 5

Inheritance & Polymorphism

1. Given the following class hierarchy diagram in Figure 1:

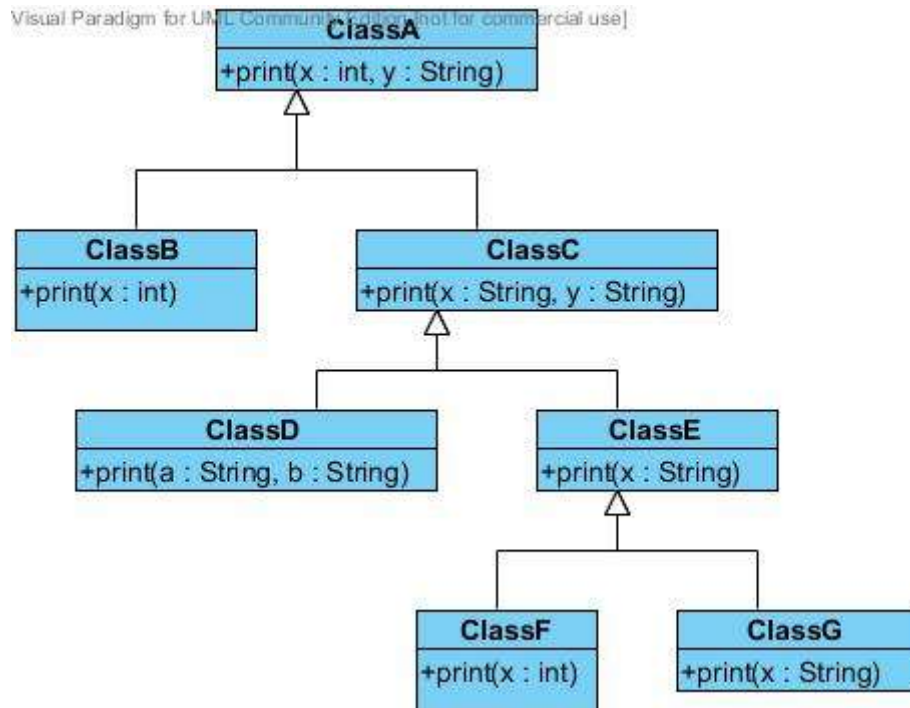**Figure 1**

Assume that `ClassF z = new ClassF();`

Which class' `print()` method will be used for each of the message below :

(a) `z.print(9)`   F   ✓

(b) `z.print(2,"Cx2002")`   A ✓

(c) `z.print("Object")`   E ✓

(d) `z.print("OODP", "Java")`   C ✓

(e) `z.print("OODP", 2002)`   Error ✓

2. Using Figure 1, and assuming all print methods just print out the contents of the its parameter values, answer the following :

(a) if the method `print(String, String)` in class `ClassC` is declared as abstract, describe what will happen and how to resolve it.

ClassC becomes an abstract class and an error occurs.

ClassD, ClassE, ClassF and ClassG have to implement print(String ,String) in their classes

ClassE needs to be declared as abstract

(b) After resolving (a), what will be the outcome of the following codes :

i.
```
ClassC  c = new ClassD();
c.print("hello","there");
```
hellothere ✓

ii.
```
ClassA  a = new ClassC();
a.print(1,"there");
```
~~1there~~   ClassC is abstract class which cannot instantiate obj

iii.
```
ClassA a  = new ClassF();
a.print("hello","there");
```
~~hellothere~~   compile error as ClassA has no print (String,String)method

(c) Assume all classes are concrete classes, what will be the outcome of the following codes :

i.
```
ClassC  c = new ClassD();
ClassE e = c;
```
compile error ✓

ii.
```
ClassB  b = new ClassE();
b.print("hello");
```
error ✓   not upcast, different family line

iii.
```
ClassA  a = new ClassF();
a.print(12,"there");
a.print(88);
```
compile error, ClassA does not have print(int) method
~~12there88~~

iv.
```
ClassA  a = new ClassC();
ClassG  g = (ClassG)a;
g.print("hello");
```
ClassC part of Class A
ClassG part of ClassC
~~hello~~   runtime error. Object is ClassC, cannot be cast to downcasting.G

v.
```
ClassA  a = new ClassC();
ClassG  g = (ClassG)a;
g.print("hello","there");
```
C to G downcasting
~~hellothere~~

vi.
```
ClassA  a = new ClassF();
ClassC  f = (ClassC)a;
f.print(88,"there");
```
88there   runtiome ok as a is ClassF object and ClassF to ClassC is upcasting

3. Figure 2 (given on the next page) lists the Java code for a Polygon class. Two subclasses, **Rectangle** and **Triangle**, are derived from the **Polygon** class.

(i)   Write the code for the **Rectangle and Triangle** subclass.

(ii)   Write a TestPolygon class to have a overloaded method **printArea**(….) which will calculate and printout the area of the polygon type passed as argument, ie printArea(**Rectangle r)** and printArea(**Triangle t).**

CE/CZ2002 OODP

(iii)   Write the **main( ) method** to demonstrate static binding of all **printArea** methods.[*Hints* : have overloaded **printArea** methods for each **Polygon** subclass].
What is the impact on the program when a new subclass of **Polygon** is introduced?

(iv)   Repeat part (ii) for dynamic binding of **printArea( )**.
[Hints : have a single printArea method, regardless of which Polygon subclass].

(v)   Modify the **Polygon** code so that any of its subclasses must include a **calArea()** member method. Suggest reason(s) why this requirement would be appropriate in this case.

This is because all polygons have an area and are different based on the polygon shape. So each polygon subclass can implement the abstract method themselves.

```java
public class Polygon {

        public enum KindofPolygon { POLY_PLAIN, POLY_RECT,
        POLY_TRIANG};
        protected String name;
        protected float width;
        protected float height;
        protected KindofPolygon polytype;

        public Polygon(String theName, float theWidth, float theHeight)
{
                    name = theName;
                    width = theWidth;
                    height = theHeight;
                    polytype = KindofPolygon.POLY_PLAIN;
        }

        public KindofPolygon getPolytype() {
                    return polytype;
        }

        public void setPolytype(KindofPolygon value) {
              polytype = value;
        }
        public String getName()  { return name; }
        public float calArea() { return 0; }
        public void printWidthHeight( ) {
              System.out.println("Width = " + width +  " Height = " +
              height);
        }
}
```

Figure 2

13

Tutorial 6

Class Diagram

1. Given the following set of classes, draw a **Class Diagram** to show the appropriate relationship between them. Add multiplicity, rolename and association name, if necessary :

    (i)   Library, LibraryResource, Book, AudioVisual, Magazine

    (ii)   Driver, Car, Wheel, Engine

    (iii)   Plane, City, Passenger, FlightTicket

    (iv)   Company, Person, Department, Job

    (v)   Product,   Inventory,   ItemStock,   Catalog,   Order,   OrderLineItem, Manufacturer
    (imagine an eCommerce system to browse through catalog to select the product/s to purchase. System will check whether there is still stocks in the inventory for the product you purchasing)

2. The following requirements describe a library system containing accounts of those users who want to access library documents. A document can be contained either directly in a library or a folder. A folder can be contained inside another folder or inside the library. Each account has its associated capability (access privilege) which provides the access levels to the different type of library items. When a user wants to access a document or a folder, his/her account's capability is checked against an access level required by the document or folder. If a user has an account, he/she can logon to the library. The user with the right capability can open, delete, and copy a folder or a document. A document can be edited also by the user. [An example of the access capability is shown Figure 1.23]

    Identify the classes you will need for the library system and draw them on a UML Class Diagram. Your Class Diagram should show clearly the relationship between classes, relevant attributes (at least one) and, multiplicity, rolename, association name, if any. You may also add in the relevant methods.

Access Capabilities	
No Access	No access permission granted
Read (R)	Read but make no changes
Write (W)	Write to file. Includes change capability
Execute (X)	Execute a program
Delete (D)	Delete a file
Change (C)	Read, write, execute, and delete. May not change file permission.
List (L)	List the files in a directory
Full Control (FC)	All abilities. Includes changing access control permissions.

Access Permissions	
Public	R – L
Group	R – X
Owner	R – W – X – D
Admins	FC
System	FC

Figure 1.23 An example of access permissions. Access permissions are applied to an object based on the level of clearance given to a subject.

ACCESS CONTROL LIST

Mary:

UserMary Directory - FullControl

UserBob Directory - Write

UserBruce Directory - Write

Printer 001 – Execute

Bob:

UserMary Directory - Read

UserBob Directory - Full Control

UserBruce Directory - Write

Printer 001 – Execute

Bruce:

UserMary Directory - No Access

User Bob Directory - Write

UserBruce Directory - Full Control

Printer 001 – Execute

Sally:

UserMary Directory - No Access

UserBob Directory - No Access

UserBruce Directory - No Access

Printer 001 - No Access

ACCESS CONTROL LIST (Continued)

Group Administrators:

    Members- Ted, Alice

    UserBruce Directory - Full Control

    UserSally Directory - Full Control

    UserBob Directory - Full Control

    UserMary Directory - Full Control

Group Printer Users:

    Members – Bruce, Sally, Bob

    UserBruce Directory – No Access

    UserSally Directory - No Access

    UserBob Directory - No Access

    PrinterDevice P1 – Print

    PrinterDevice P2 – Print

    PrinterDevice P3 – Print

Tutorial 7

Design Principles

1. Look at Figure 4 below. To cater to more type of accounts, the Account class is to be modified to check for the type of accounts and provide the appropriate balance in the getAvailableFunds() method. However, the initial Account class has been working fine and it is advised that it should not be modified.

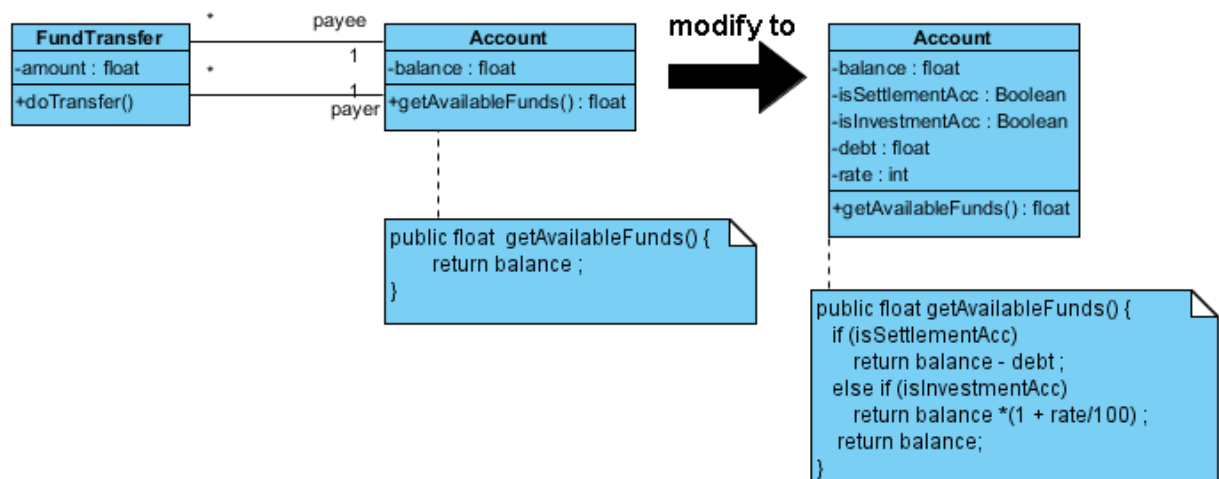   Propose a design principle to apply and show the new design in UML Class Diagram.

Figure 3

2. Look at Figure 5 below. The BankingWebService uses Account class toXML() and fromXML() methods to display and update the account details respectively.

   a. Identify the responsibilities of the Account class.

   b. Apply the Single Responsibility Principle to the class and show the new design in UML Class diagram.

   c. Explain what needs to be done if either of the deposit or withdraw methods is modified.

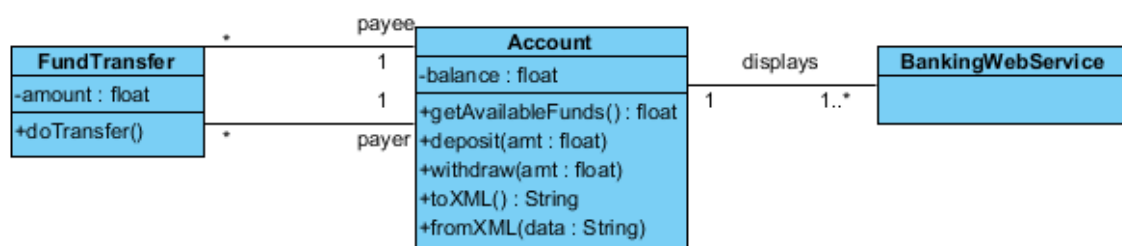   d. How can you improve (c)? Identify the principle used and show the solution in UML Class Diagram.

Figure 4

3. Look at Figure 6 below. Using Liskov Substitution Principle (LSP), comment on the design.


```
client provide the recipient list for
MessageBroadcaster or its subtypes

It will display the number of
messages sent.
```

```
public int send(Recipient [] recipients) {
    int noOfRecipient;
    if (recipients == null)
        return 0;
    else {
     // continue to send to all recipients
     // and count successful send
     // using noOfRecipient
    }
    return noOfRecipient ;
}
```

Client

uses

Recipient

MessageBroadcaster
#message : String
+send(recipients : Recipient []) : int

EmailBroadcaster
+send(recipients : Recipient []) : int

```
public int send(Recipient [] recipients) {
    if (recipients == null)
        throw new IllegalArgumentException() ;
    else {
     // continue to send to all recipients
    }
    return 0 ;
}
```
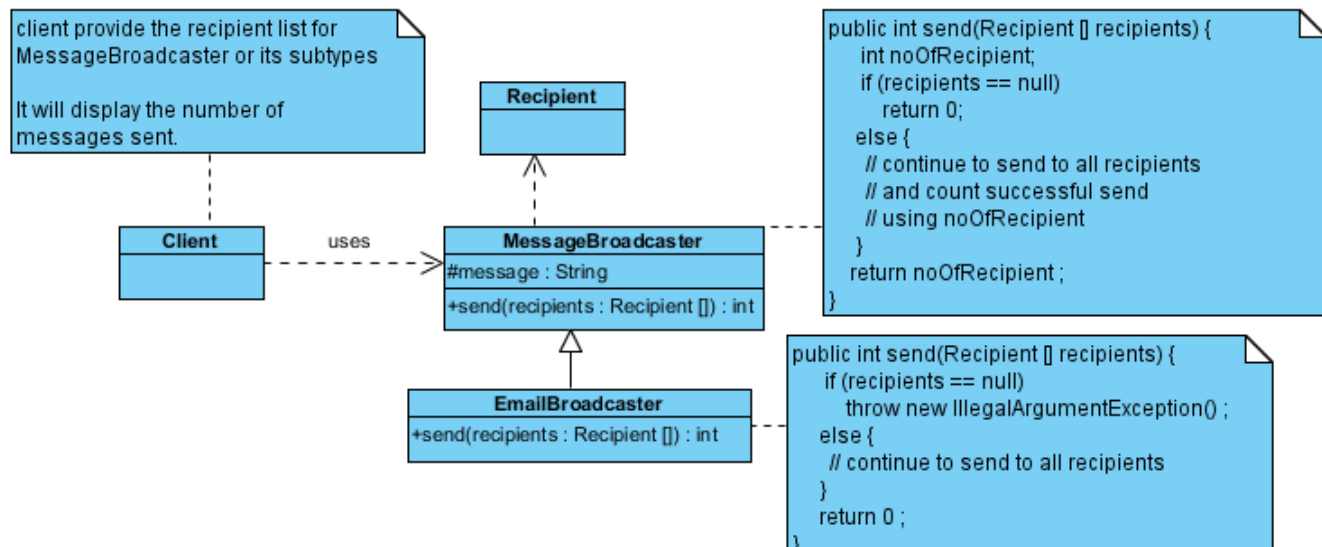
Figure 5

17

Tutorial 8

Object-Oriented in C++

1. Convert the following Bubble sort program in *C language* code to *C++ language* code by :
   a. Identify the line number of the code to be changed
   b. Directly replace with the C++ language syntax.
   c. Write the code as a C++ <u>class</u>.

```
1   #include<stdio.h>              #include <iostream>
2                                  using namespace std;
3   void bubble(int a[],int n)
4   {
5       int i,j,t;
6     for(i=n-2;i>=0;i--)
7       {
8       for(j=0;j<=i;j++)
9         {
10          if(a[j]>a[j+1])
11              {
12            t=a[j];
13            a[j]=a[j+1];
14            a[j+1]=t;
15          }
16       }
17     }
18   }//end function.
19
20   int main()
21    {
22       int a[100],n,i;
23                           cout<< "\n\n Enter number of Integer elements to be sorted: "<<endl;
24       printf("\n\n Enter number of Integer elements to be sorted: ");
25       scanf("%d",&n);              cin >> n;
26
27       for( i=0;i<=n-1;i++)
28       {                  cout << "\n\n Enter integer value for element no. << i+1 << ":" <<endl;
29          printf("\n\n Enter integer value for element no.%d : ",i+1);
30          scanf("%d",&a[i]);        cin>> a[i];
31       }
32
33       bubble(a,n);
34                           cout<< "\n\n Finally sorted array is: " <<endl;
35       printf("\n\n Finally sorted array is: ");
36       for( i=0;i<=n-1;i++)
37           printf("%d  ",a[i]);   cout << a[i] <<" " <<endl;
38     } //end program.
```

2. Referring to Tutorial 5 Question 3(v) and its solution, implement the codes for :
   a. the `Polygon` class,
   b. its subclasses `Rectangle` and `Triangle` class AND
   c. a `printArea` function demonstrating dynamic binding of the `calArea` implementation.

Tutorial 9:

Polymorphism in C++

1. Debug and run the following program.
   a) What will be the output?
   b) Write an operator overloaded method/function to add 2 objects of class A together and return the result as class A object.

```
1   #include <iostream.h>
2                               using namespace std;
3   class A {
4    protected :
5          int a,b;                      .
6    public :
7          A(int x=0, int y) {          - 0
8                a = x ;              constructor
9                b = y ;
10         }
11         virtual void print() ;
12            int getA(){return a;}
13
14  };
15  class B: public A {
16   private:
17           float p,q ;        .
18   public :
19         B(int m, int n, float u, float v) {
20               p = u ;                         constructors
21               q = v ;
22         }
23         B() { p = q = 0 ; }
24         void input(float u, float v) ;
25         virtual void print(float) ;
26  };
27  void A::print(void) {
28         cout << "A values: " << a << " " << b << "\n" ;
29  }
30  void B::print(float) {
31         cout << "B values : " << u << " " << v << "\n" ;
32  }
33  void B::input(float x, float y) {
34         p = x ;   q = y ;
35  }
36
37  int doubleIt(A a) { return a.a * a.a ; }
38                               getA()   getA()
39  main() {
40    A a1(10,20), A*ptr ;
41    B b1;
42    b1.input(7.5, 3.142) ;
43
44    ptr = &a1 ;          A values: 0 20
45    ptr->print() ;       A values: 0 0
46    ptr = &b1;
47    ptr->print() ;
    }
```

2. Run the following program. What will be the output?

```
1  #include <iostream>
2  using namespace std;
3  class BC {
4    public :
5         void show(void) { cout << " \n I am in base class..";  }
6  };
7  class DC :public BC {
8    public :
9         void show(void) { cout << " \n I am in derived class..";  }
10 };
11 int main() {
12
13   BC* ptr1 ;
14   DC dobj;
15   ptr1 = &dobj;
16   ptr1->show() ;
17 }
```

no virtual

I am in the base class