SC2002:
OBJECT ORIENTED DESIGN & PROGRAMMING

Tutorials

CCDS
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 *Python 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.

```
1   def bubble(a, n):
2       for i in range(n-2, -1, -1):
3           for j in range(i+1):
4               if a[j] > a[j+1]:
5                   a[j], a[j+1] = a[j+1], a[j]
6
7   def main():
8       a = []
9
10      n = int(input("\n\nEnter number of integer elements to be sorted: "))
11
12      for i in range(n):
13          value = int(input(f"\n\nEnter integer value for element no. {i+1}:
14  "))
15          a.append(value)
16
17      bubble(a, n)
18
19      print("\n\nFinally sorted array is: ", end="")
20      for i in range(n):
21          print(a[i], end="  ")
22
23  if __name__ == "__main__":
24      main()
25
```

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
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 instance variables and 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.

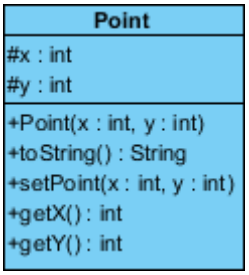
a) Discuss the design of the VendingMachine class and how it can be improved.
b) What will be a relevant class to relate to the VendingMachine class?
c) Suggest how the application can be re-designed to involve the class in (b)?
d) [Optional] Implement your design.

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 :



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

Write the code in Java.

Create a class Circle to extend from the Point class. The Circle class is to have the following methods: **`setRadius`**, **`getRadius`** and **`area`**. Reuse whatever you can from the Point class.

Create a class `Cylinder` to extend from the any of the classes above. The `Cylinder` class is to have the following methods: **`setHeight`**, **`getRadius`**, **getHeight**, **`area`** and **`volume`**.

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.

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.");
      }
```

```
    }
    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:

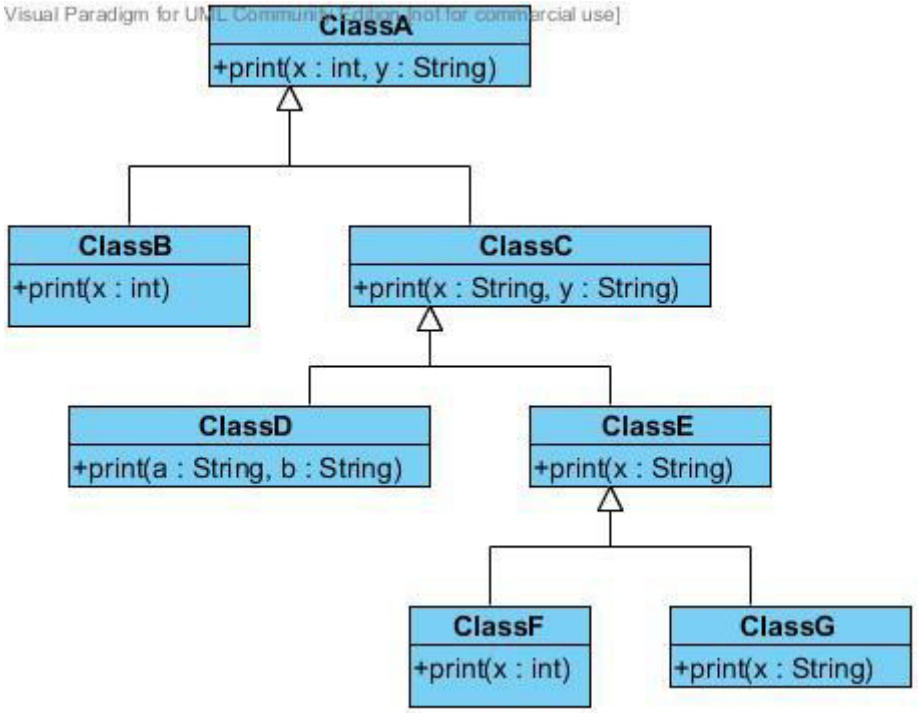


Figure 1

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

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

(a) `z.print(9)`

(b) `z.print(2,"Cx2002")`

(c) `z.print("Object")`

(d) `z.print("OODP", "Java")`

(e) `z.print("OODP", 2002)`

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.

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

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

ii.
```
ClassA  a = new ClassC();
a.print(1,"there");
```

iii.
```
ClassA a  = new ClassF();
a.print("hello","there");
```

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

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

ii.
```
ClassB  b = new ClassE();
b.print("hello");
```

iii.
```
ClassA  a = new ClassF();
a.print(12,"there");
a.print(88);
```

iv.
```
ClassA  a = new ClassC();
ClassG  g  = (ClassG)a;
g.print("hello");
```

v.
```
ClassA  a = new ClassC();
ClassG  g  = (ClassG)a;
g.print("hello","there");
```

vi.
```
ClassA  a = new ClassF();
ClassC  f  = (ClassC)a;
f.print(88,"there");
```

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)**.

(iii) Write the **main**( ) function 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.

```
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

Tutorial 6

Class Relationship & Model to Code

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)

   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]

2. 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

2.

Tutorial 7

UML Sequence Diagram

1. The **UML Sequence Diagram** in Figure Q shows the objects interaction of the "Cash Check" scenario flow. Using the details depicted in the diagram, write the preliminary codes of the JAVA classes for all the participating objects shown. You may make appropriate assumptions on the method parameters and return types.

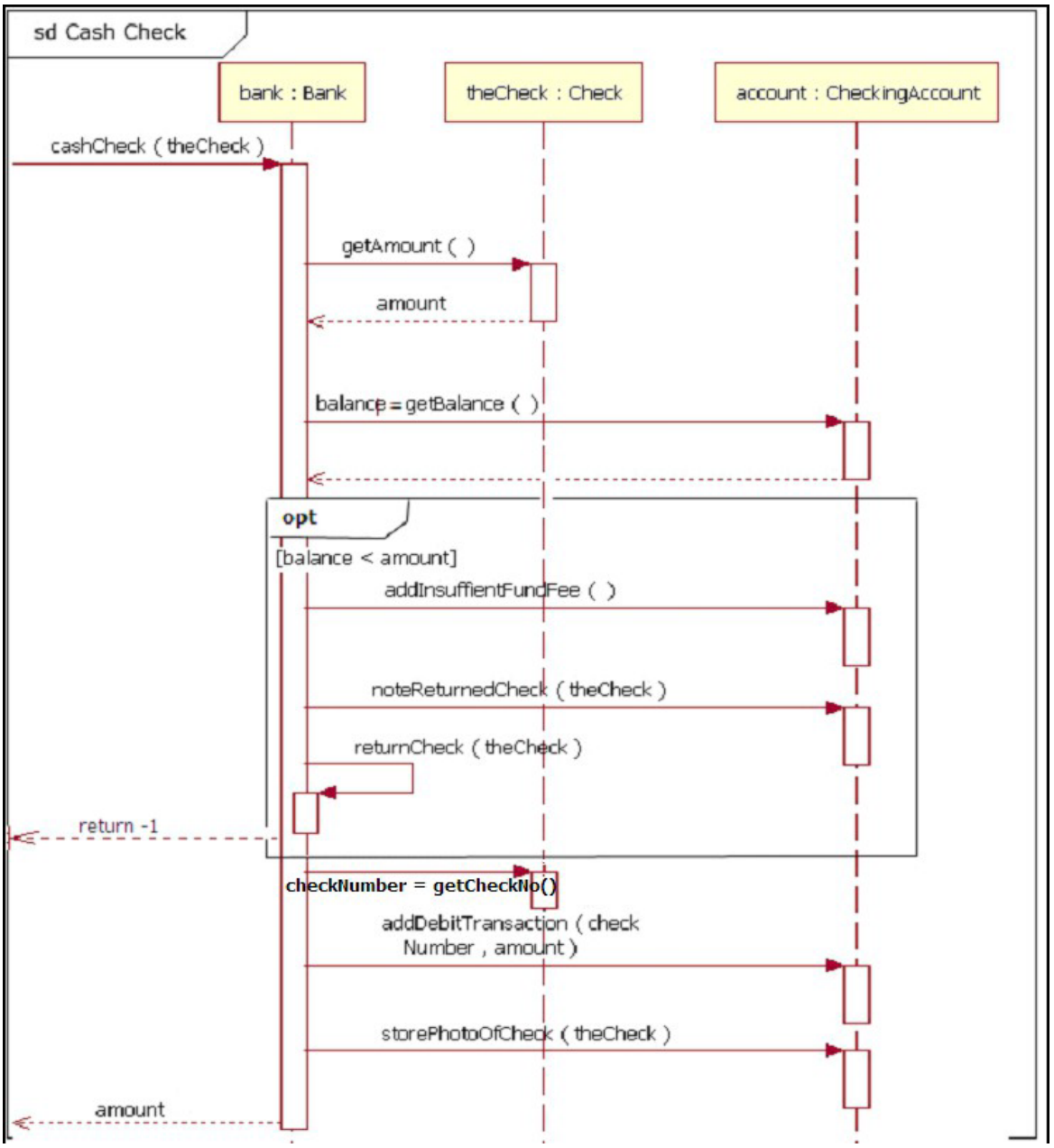

sd Cash Check
bank : Bank
theCheck : Check
account : CheckingAccount
cashCheck (theCheck)
getAmount ( )
amount
balance = getBalance ( )
opt
[balance < amount]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)
return -1
checkNumber = getCheckNo()
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
amount

Figure Q

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]

2. Draw the **UML Sequence Diagram** to show the flow of a *user accessing its library items*.

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 8

Design Principles

a. 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.

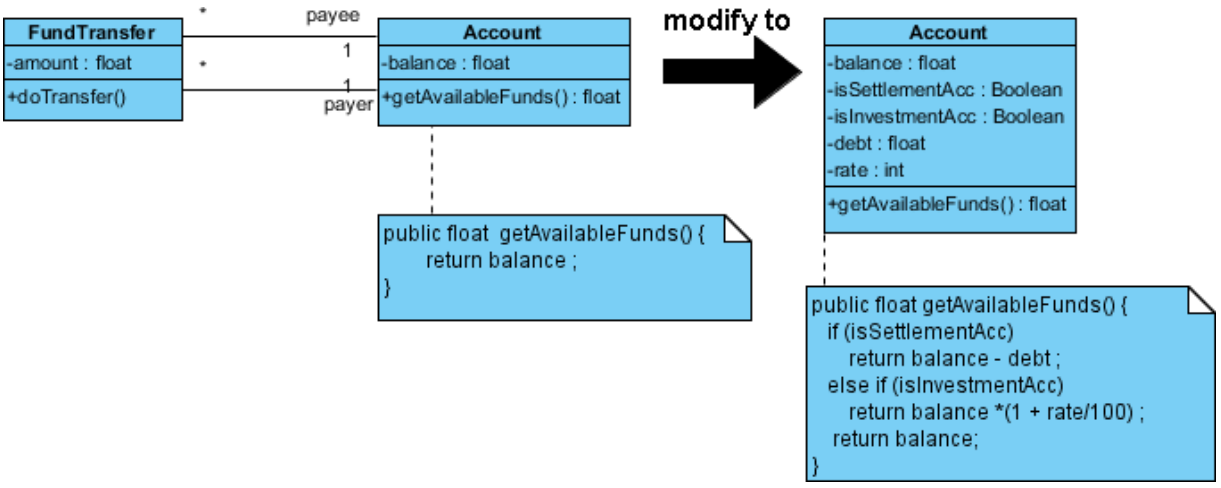


Figure 4

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

i. Identify the responsibilities of the Account class.

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

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

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

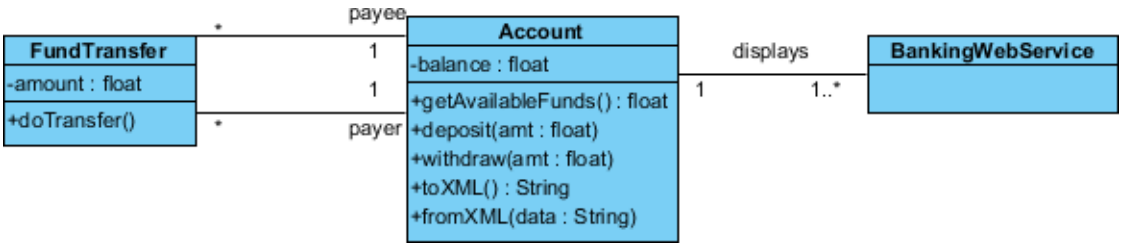


Figure 5

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

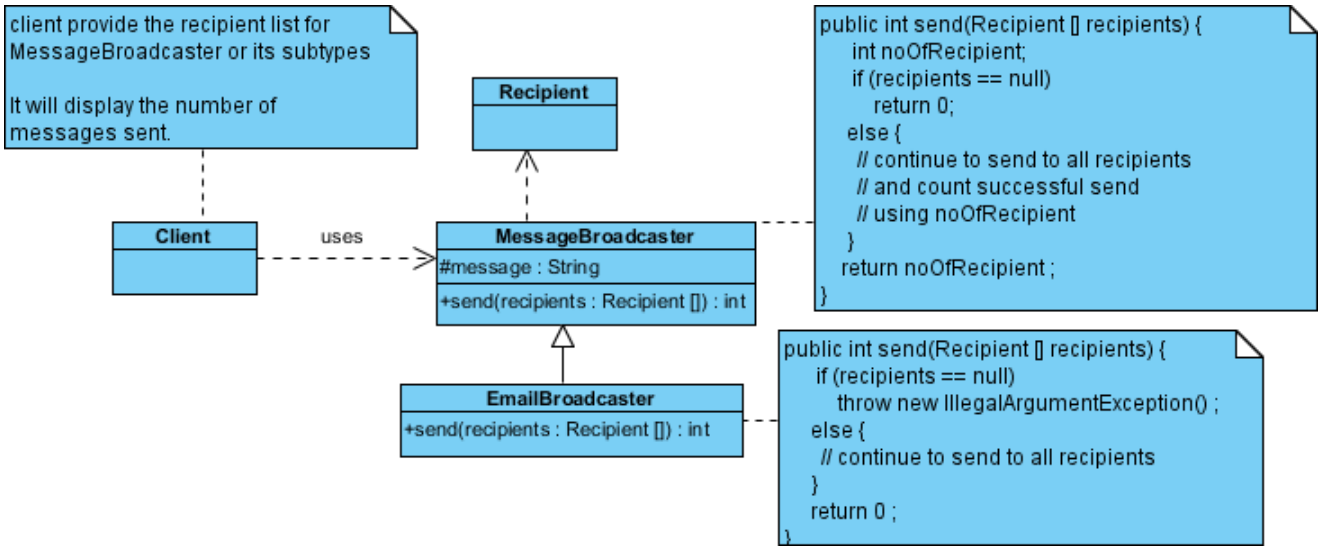


Figure 6

Tutorial 9

Mastering Java Enhancements: Generics, Streams, Lambdas, and More

Q1

a) Write a generic method printArray that takes an array of any type and prints its elements. Test your method with arrays of Integer, String, and Double.
b) What happens if you try to pass a primitive array (e.g.,int[]) to the printArray method? Why?
c) How can you modify the method to handle primitive arrays?

Q2

a) Use the Collections utility class to perform the following operations on a List<Integer>:
    - Sort the list in descending order.
    - Shuffle the list.
    - Find the maximum and minimum values in the list.
b) How does the sort method handle null values in the list?
c) What is the difference between Collections.sort() and List.sort()?

Q3

a) Given a list of strings, use the Streams API to:
    - Filter out all strings with a length less than 5.
    - Convert the remaining strings to uppercase.
    - Collect the results into a new list.
b) How would you modify the code to count the number of strings that meet the filtering criteria?
c) What is the difference between map() and flatMap() in the Streams API?

Q4

a) Rewrite the following anonymous class using a lambda expression:

```
Runnable r = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello, World!");
    }
};
```

b) Can all functional interfaces be replaced with lambda expressions? Why or why not?
c) What are the benefits of using lambda expressions over anonymous classes?

Q5

a) Given an array of integers, use the enhanced for loop to calculate the sum of all even numbers in the array.
b) How does the enhanced for loop differ from the traditional for loop?

c) Can you use the enhanced for loop with custom objects? If so, how?

Q6

a) Create an interface Vehicle with a default method start() that prints "Vehicle started." Then, create a class Car that implements Vehicle and overrides the start() method to print "Car started."
b) What happens if a class implements two interfaces with the same default method? How can this conflict be resolved?
c) Why were default methods introduced in Java 8?

Q7

a) Rewrite the following switch statement using the enhanced switch syntax:

```
int day = 3;
String dayType;
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        dayType = "Weekday";
        break;
    case 6:
    case 7:
        dayType = "Weekend";
        break;
    default:
        dayType = "Invalid day";
}
```

b) What are the advantages of using the enhanced switch over the traditional switch statement?
c) Can the enhanced switch be used with non-primitive types (e.g., String)? If so, how?