Lab 07 Review, Inheritance and More

Q1-6 MC Questions - finish on **Canvas**
Q7 Programming - submit to PASS

Q1-Q6 Read the following notes. Then answer the questions on **Canvas** (MC question).

Review

Visibility (public / protected / private):

- When we implement or redefine a method in the subclass, it must be at least as "visible" as one in the superclass.
- A subclass cannot access the private members in the superclass.

The static keyword

Used to denote fields and methods that belong to a class (but not to any particular object).

The abstract keyword

The **abstract** keyword is applied for **classes** and **nonstatic methods**:

- When applied for a nonstatic **method**: means that we intend to provide **no implementation**; and the implementation will be provided in concrete subclasses.
- When applied for a **class**: means that the class may or may not include abstract methods. Abstract classes **cannot be instantiated** (ie. cannot be used to instantiate any object), but they **can be subclassed**.

- **abstract** is NOT for fields (no matter static or nonstatic)
- **abstract** is NOT for constructors or static methods

Polymorphism – An object variable can refer to different actual types. [compile time checking]
Superclass Superclass and subclass which are concrete
E.g., An object variable (of type A) can refer to objects of various actual types, including type A and its sub-types.

Dynamic Binding – Automatically select the appropriate non-static method. [runtime checking]
Not field!

Q1 Use of `abstract`

Read the code below. Which lines contain invalid code? Explain.

```
abstract class A
{
   public int p1;              //line 1
   public abstract int p2;     //line 2

   public void x1() {}         //line 3
   public void x2();           //line 4

   public abstract void y1() {}  //line 5
   public abstract void y2();    //line 6

   public abstract static void z1();  //line 7
}
```

Note: Answer choices are given on canvas! Test the program! (Download from course web!)

Q2 static and non-static fields and methods

Read the code below.

(a) For line 1-4, which line(s) contain(s) invalid code? Explain.

(b) Assume that now each error in line 1-4 is removed.
For line 5-12, which line(s) contain(s) invalid code? Explain.

```
class A {

   private int i;
   private static int j;

   public static void f1() {
      i++;   //line 1
   }

   public void f2() {
      i++;   //line 2
   }

   public static void f3() {
      j++;   //line 3
   }

   public void f4() {
      j++;   //line 4
   }

}


public class Main_Lab07Q2
{
   public static void main(String[] args)
   {
      A obj = new A();

      obj.f1(); //line 5
      obj.f2(); //line 6
      obj.f3(); //line 7
      obj.f4(); //line 8

      A.f1();   //line 9
      A.f2();   //line 10
      A.f3();   //line 11
      A.f4();   //line 12
   }
}
```

Q3 Visibility in inheritance

Read the code below. Which lines contain invalid code? Explain.

```
abstract class A {
   public int i;
   protected int j;
   private int k;

   public static void print1() {}
   public void print2() {}
}

class B extends A
{
   protected static void print1() //line 1
   {
      System.out.println(i); //line 2
      System.out.println(j); //line 3
      System.out.println(k); //line 4
   }

   protected void print2() //line 5
   {
      System.out.println(i); //line 6
      System.out.println(j); //line 7
      System.out.println(k); //line 8
   }
}
```

Q4 Polymorphism and Dynamic binding

Read the code below. Which lines contain invalid code? Explain.

```
abstract class A {
   public abstract void fi();
}

class B extends A {
   public void fi() {}
   public void fj() {}
}

public class Main_Lab07Q4
{
   public static void main(String[] args)
   {
      A a;           //line 1
      a = new A();   //line 2
      a.fi();        //line 3  (Assume any error(s) in line1-2 are removed)

      B b;           //line 4
      b = new B();   //line 5
      b.fj();        //line 6

      B b1 = a;      //line 7
      A a1 = b;      //line 8
      b.fj();        //line 9
      a1.fj();       //line 10
   }
}
```

Q5 Inheritance, Polymorphism, Dynamic Binding, static/non-static

What is the output of the following program? (It has no compile-time or run-time error)

```
class A {
    public int i; //JAVA: default initialization for numeric fields is 0
    public static int j; //JAVA: default initialization for numeric fields is 0
    A() {i++;j++;}
    public void fi() {i++;}
    public void fj() {j++;}
    public static void sj() {j++;}
}

class B extends A {
    public int i;
    public static int j;
    public void fi() {i++;}
    public void fj() {j++;}
    public static void sj() {j++;}
}

public class Main_Lab07Q5
{
    public static void main(String[] args)
    {
        A a;
        a = new A();
        a.fi();
        a.fj();
        a.sj();

        B b;
        b = new B();
        b.fi();
        b.fj();
        b.sj();
        System.out.println(a.i+" "+a.j+" "+b.i+" "+b.j);

        A a1 = b;
        a1.fi();
        a1.fj();
        a1.sj();
        System.out.println(a.i+" "+a.j+" "+b.i+" "+b.j);
    }
}
```

Outut:

Q6 Inheritance, Polymorphism, Dynamic Binding, super

What is the output of the following program? It has no compile-time or run-time error.

```
class P {
    int k;
    public P() {k=1;}
    public void triple() {k=3*k;}
    public void print() {
        triple();
        System.out.println("In P: "+k);
    }
}

class Q extends P {
    int k;
    public Q() {k=10;}
    public void triple() {k=3*super.k;}
    public void print() {
        this.triple();
        System.out.println("In Q: "+k);
    }
}

class R extends Q {
    int k;
    public R() {k=100;}
    public void triple() {k=3*k;}
    public void print() {
        super.print();
        System.out.println("In R: "+k);
    }
}

public class Main
{
    public static void main(String[] args)
    {
        P x1=new P();
        x1.print();

        Q x2=new Q();
        x2.print();

        R x4=new R();
        x4.print();
    }
}
```

Output:

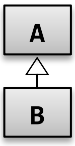
Lab07 - Progress Check

[Exit-test] Read the following note. Then write down the output for the two programs below.

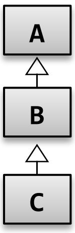
this and ***super*** work in very different way:

- When "this" is used in a statement: "this" means the runtime object (dynamic decision at runtime)
- When "super" is used in a statement: "super" means the parent class of that statement's class (confirm at compile time)

```
Program I

class A {
    public int value=1;
}

class B extends A {
    public int value=2;
}

public static void main(String [] args)
{
    A x = new B();

    System.out.println(x.value);      //output: ____

    System.out.println(((B)x).value); //output: ____
}
```



```
Program II

class A {
   public void print() { System.out.println("A");}
}

class B extends A {
   public void print() { System.out.println("B");}
   public void printSuper() {super.print();}
}

class C extends B {
}

public static void main(String [] args)
{
   A a = new B();
   a.print();               //output: _________

   (new B()).printSuper();  //output: _________
   (new C()).printSuper();  //output: _________
}
```



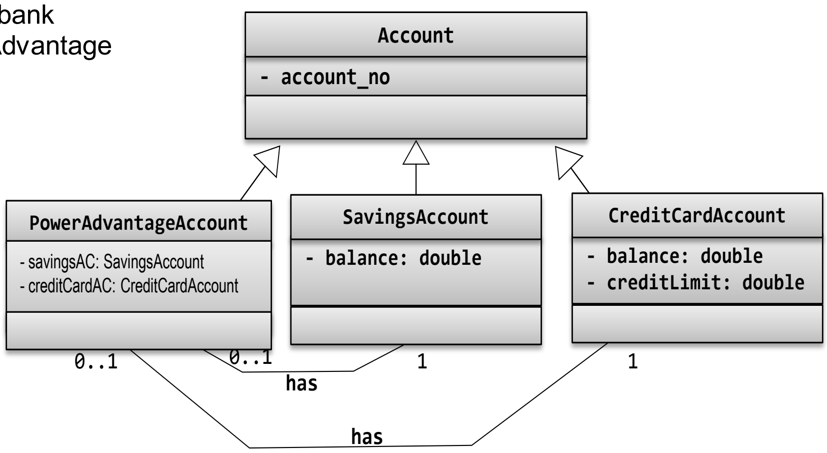
Anything unclear? Ask.

Q7. Programming

ABC bank manages3 different types of bank accounts: Savings, Credit card, PowerAdvantage account

A PowerAdvantage account is actually a pair of a savings account and a credit card account.

You are to complete a program which gets account details from a file, and lets the user search an account.



Sample file contents:

Each line stores the data of one account:

Savings account: a/c number, balance
(a/c number starts with 0-5)

Credit card account: a/c number, balance, credit limit
(a/c number starts with 6-8)

PowerAdvantage account: a/c number, 2 sub-account numbers (exist already, 1st is savings, 2nd is credit card)
(a/c number starts with 9)

a1.txt

```
0123456789 5000
1111222222 6000
3333334444 1000
6666888888 8000 20000
9999123456 0123456789 6666888888
```

Sample rundown #1:

```
Please input the file pathname: c:\a1.txt

Input an account number to search: 1122334455

[Result]
The account is not found.
```

Sample rundown #2:

```
Please input the file pathname: c:\a1.txt

Input an account number to search: 0123456789

[Result]
Savings A/C Number: 0123456789 Balance: $5000.00
```

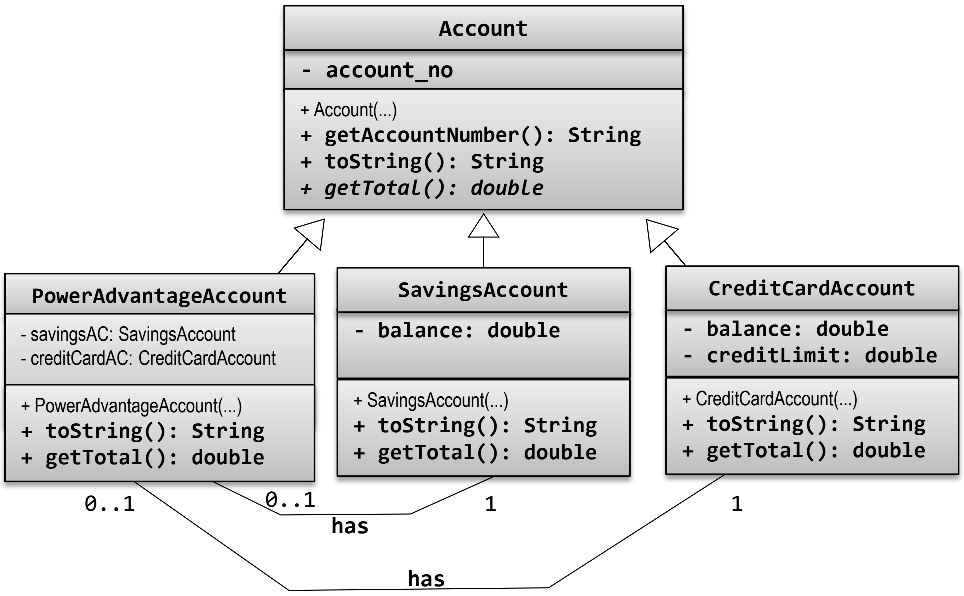
Sample rundown #3:

```
Please input the file pathname: c:\a1.txt

Input an account number to search: 6666888888

[Result]
Credit Card A/C Number: 6666888888 Balance: $8000.00 Credit limit: $20000.00
```

Sample rundown #4:

```
Please input the file pathname: c:\a1.txt

Input an account number to search: 9999123456

[Result]
Power Advantage A/C Number: 9999123456 Balance: $-3000.00
 1.Savings A/C Number: 0123456789 Balance: $5000.00
 2.Credit Card A/C Number: 6666888888 Balance: $8000.00 Credit limit: $20000.00
```

Details in the classes:



Your tasks: Download the given .java files and complete the classes step by step. Submit your work to PASS.

1. Finish the Account class: public abstract class Account

   Instance field: private String account_no;

   Methods:
   public Account(String ano) {..} //constructor
   public String getAccountNumber(){..}
   public abstract double getTotal(); //This abstract method will be implemented in subclasses.
   public String toString() {..} //Return a string like: "Bank A/C Number: 0123456789" ← **add @Override**

2. Finish the SavingsAccount class:

   Instance field: private double balance;

   Methods:
   public SavingsAccount(String ano, double bal) {..} //constructor
   public double getTotal() {..} //Implement the abstract method in Account ← **add @Override**
   public String toString() {..} //Return a string like the output in rundown #2 ← **add @Override**

3. Finish the CreditCardAccount class:

   Instance fields: private double balance;
   private double creditLimit;

   Methods:
   public CreditCardAccount (String ano, double bal, double climit) {..} //constructor
   public double getTotal() {..} //Implement the abstract method in Account ← **add @Override**
   public String toString() {..} //Return a string like the output in rundown #3 ← **add @Override**

4. Finish the PowerAdvantageAccount class:

   Instance fields: private SavingsAccount savingsAC;
   private CreditCardAccount creditCardAC;

   Methods:
   public PowerAdvantageAccount (String ano, SavingsAccount sa, CreditCardAccount cr) {..} //constructor
   public double getTotal() {..} //Implement the abstract method in Account ← **add @Override**
   public String toString() {..} //Return a string like the output in rundown #4 ← **add @Override**

5. Finish the Main class:

   Methods:
   private static Account findAccount(ArrayList<Account> list, String account_no) {..} //searching a/c
   public static void main(String [] args) {..}

-- END --