COMP10062: Final Exam Review

Sam Scott, Mohawk College, Winter, 2018.

This is a compendium of questions taken from courses with content similar to COMP10062. Keep in mind that the final exam is cumulative, so you should also look at review questions from the last two tests.

Content not covered here that you should expect on the exam:

- Creating custom exceptions
- Mouse listeners
- Interfaces

A. Knowledge and Understanding

1. Write a single line of code that will throw an `IllegalArgumentException` containing the message "Nice try". [1]

```
throw new IllegalArgumentException("Nice try");
```

2. Write code to call a method named `testMethod()`. This method might throw an `IllegalArgumentException`. Include code to catch that exception and print the message contained in that exception to `System.err`. [4]

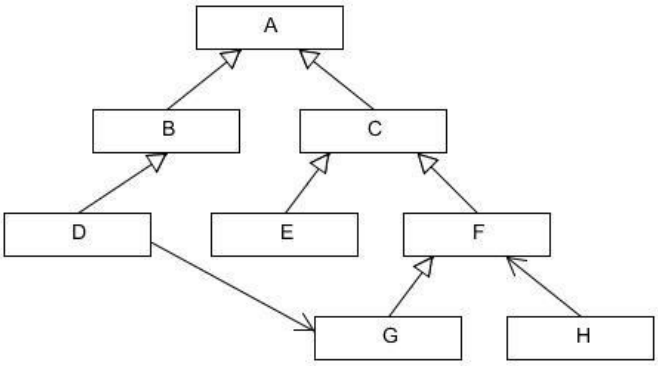
```
try
{
    testMethod();
}
catch(IllegalArgumentException e)
{
    System.err.println(e.getMessage());
}
```

3. Write code to create an empty class named `Child` that inherits all its methods and instance variables from a class named `Parent`. [1]

```
public class Child extends Parent { }
```

4. Given the UML diagram and the declaration for the variable `c` below, put a check mark beside all the assignments that are legal and an X beside those that are illegal. [3]

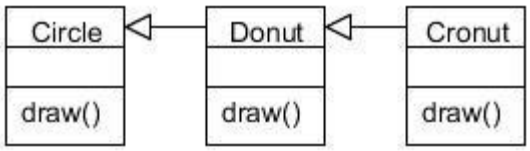
`C c;` ← This is the variable declaration



`c = new A();`	not legal
`c = new B();`	not legal
`c = new C();`	legal
`c = new D();`	not legal
`c = new E();`	legal
`c = new F();`	legal
`c = new G();`	legal
`c = new H();`	not legal

5. Consider the UML diagram and the code below. When the code is run, how many calls will there be to a draw method, and which draw method(s) will be called in each case? [3]

```
Circle c = new Donut();
c.draw();
if(c instanceof Cronut)
    ((Cronut)c).draw();
if(c instanceof Donut)
    ((Donut)c).draw();
```



Two calls to `draw()`. They both call the Donut `draw()` method.

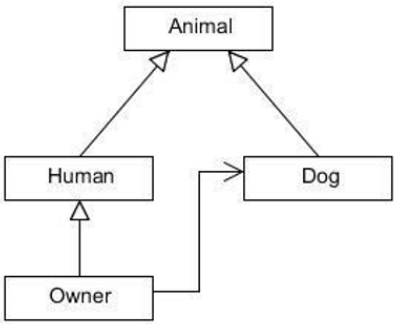
6. Write the code necessary to create an array and fill it with 1000 objects of type Donut (see the previous question for the Donut class). [3]

```
Donut[ ] d = new Donut[1000];
for(int i = 0; i < d.length; i++)
    d[i] = new Donut( );
```

7. Write code to declare an abstract class named `ABS` that contains an abstract method named `absm`. that will return a String and has no parameters[2]

```
public abstract class ABS
{
    abstract String absm( );
}
```

8. Write the minimum amount of code necessary to implement just the Owner class as shown at right. Assume the other classes have already been implemented. [3]

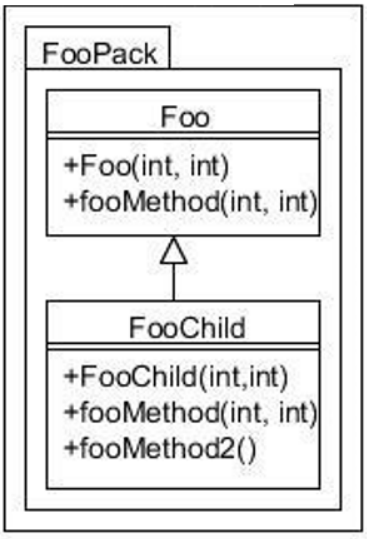



```
public class Owner extends Human
{
    Dog dog = new Dog();
}
```

9. Write the code for an abstract class named "Abs" that contains an abstract method called "foo". The abstract method has two integer parameters and returns a String. [3]

```
public abstract class Abs
{
    abstract String foo(int a, int b);
}
```

10. Questions a through c all refer to the UML diagram shown at right.



    a. What are the declared and actual types in the line of code below? [1]

    ```
    Foo x = new FooChild(0,0);
    ```

    Declared: ___Foo_____

    Actual: _____FooChild_____

    b. Write a line of code that calls `fooMethod2()` for the object stored in the variable x above. [1]

    ```
    ((FooChild)x).fooMethod2();
    ```

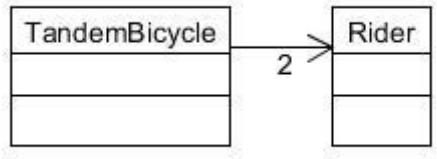
    c. What class does Foo extend? [1]

    ```
    Object
    ```

11. Write Java code to print “1” to System.out if the variable “z” holds an object of type MyType and “2” if it holds an object of the type YourType. Otherwise, print nothing. [3]

```
if(z instanceof MyType)
    System.out.println("1");
else if(z instanceof YourType)
    System.out.println("2");
```

12. Add the minimum amount of code necessary to the classes below in order to implement the relationship in the class diagram shown on the right. You can assume that each class is defined in its own file. [3]




```
public class TandemBicycle
{
    Rider r1, r2;
    //Rider[] riders;  ← this is also acceptable
    //creating Rider objects is optional
}

public class Rider
{
    //1 point for no code here
}
```

13. Explain the difference between checked and unchecked exceptions. Give one example of each type of exception. [4]

A checked exception must be either caught or be declared to be thrown by adding the “throws” declaration to the method. An unchecked exception doesn’t need either.

InputMismatchException is an example of an unchecked exception.

FileNotFoundException is an example of a checked exception.

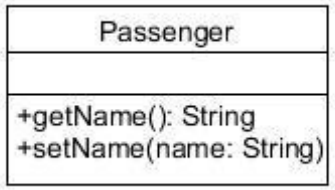
14. Write a single line of code in the method below so that it prints the value of both variables named `x` to Standard Output. [2]

```
public class Exam {
      private int x = 5;

      public void foo(int x){

            System.out.println(x + " " + this.x);
                  Put your code on this line
      }
}
```

15. Write a declaration for a private instance variable of type double named `myconstant`. Write the declaration so that you could set a value for `myconstant` in the constructor, but after that it could not be changed. [2]

```
private final double myconstant;        // any type will do
```

16. Suppose you have a variable named `passengers` which stores an array of objects of type `Passenger` (see class diagram to the right). Write a single line of code that will print the output of the `getName()` method for the first `Passenger` object in the `passengers` array. [3]




```
System.out.println(passengers[0].getName());
```

17. Using the same `passengers` variable from the last question, write an enhanced for loop to set the name of every `Passenger` object in the array to "Sally". [4]

```
for(Passenger p : passengers)
   p.setName("Sally");
```

18. Write a single line of code that will print the ASCII/UNICODE value for the third character of a String variable named `s`. [3]

```
System.out.println((int) s.charAt(2));
```

19. I have a method named `badmethod()`. It takes no arguments and has no return value but when it's called it might throw a special kind of exception called `BadMethodException`. Write code that will call `badmethod` and print "ERROR" to standard output if it throws the `BadMethodException`. [5]

```
try
{
    badmethod();
}
catch(BadMethodException e)
{
    System.out.println("ERROR");
}
```

20. What do you have to add to the constructor shown here to force a call the no-arg constructor of its parent class? [1]

```
public class MyClass extends MyOtherClass
{
      private int x,y,z;

      public MyClass(int x, int y, int z)
      {
            this.x = x;
            this.y = y;
            this.z = z;
      }
}
```

Nothing

21. Give examples of two different ways to use the `super` keyword. Explain what each use of the `super` keyword does. [3]

super( ) ← calls a constructor from the parent class (can only be used as the first line of a constructor).

super.myMethod(); ←calls a method from the parent class (use this when you have an override of myMethod and you want to call the one from the parent class).

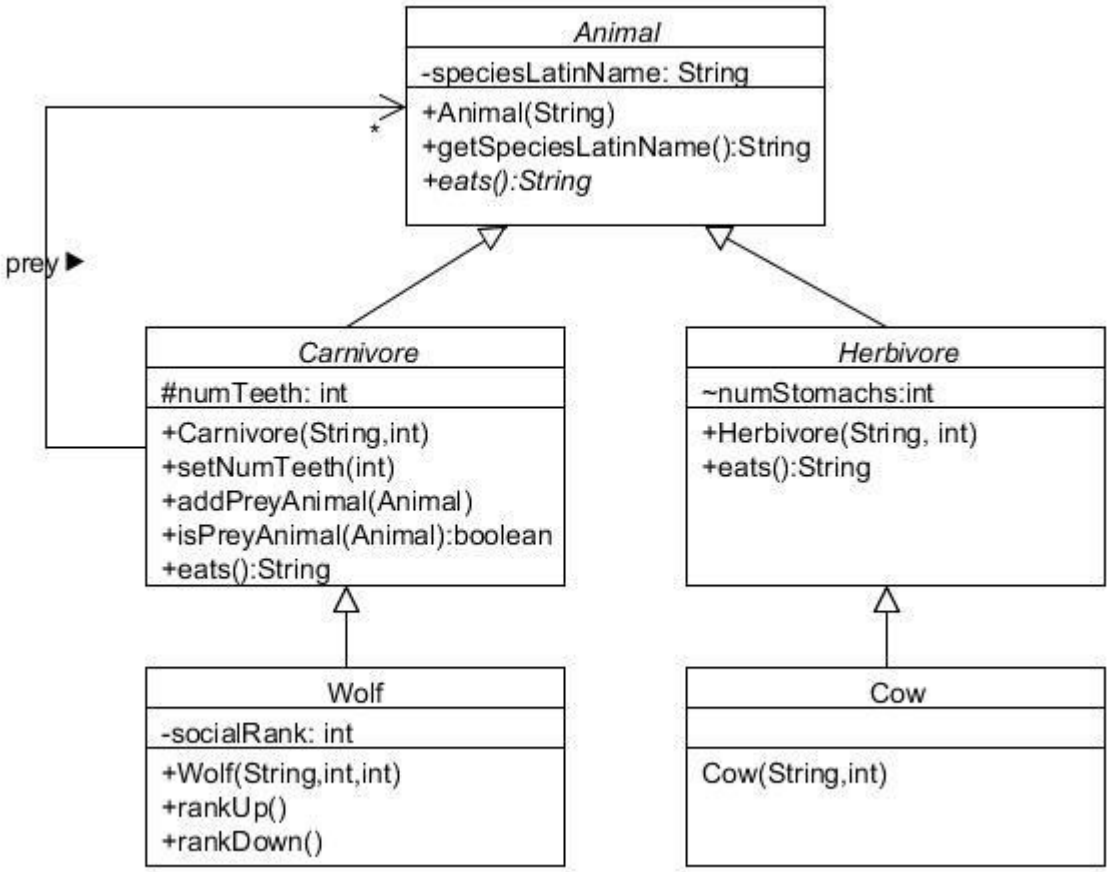
B. Application

22. Consider the UML diagram below. Your job is to write the minimum amount of code to implement everything in this diagram exactly as shown and according to the principles of good object oriented design. [20]

    Here are some implementation notes:

    - `Carnivores` eat animals and `Herbivores` eat plants
    - Note that some class names and methods are in italics.
    - A `Carnivore` contains a set of prey animals (i.e. animals that it eats)
    - A prey animal can be added to the list using the `addPreyAnimal()` method.
    - You can check if a given animal is prey for a given `Carnivore` by calling `isPreyAnimal()`.
    - A `Wolf` has a numeric social rank that can be increased or decreased by 1.

NOTE for COMP10062 students: Treat the # and ~ symbols as if they were – (private).



```
public abstract class Animal
{
    private String speciesLatinName;

    public Animal(String name){ speciesLatinName = name; }
    public String getSpeciesLatinName() { return speciesLatinName; }
    public abstract String eats();
}

public abstract class Herbivore extends Animal
{
    private int numStomachs;

    public Herbivore(String name, int stomachs)
    {
          super(name);
          numStomachs = stomachs;
    }

    public String eats() { return "plants"; }
}


public class Cow extends Herbivore
{
    public Cow(String name, int stomachs)
    {
          super(name, stomachs);
    }
}
```

```
public abstract class Carnivore extends Animal
{
    private int numTeeth;
    private ArrayList<Animal> prey;

    public Carnivore(String name, int teeth)
    {
          super(name);
          setNumTeeth(teeth);
          prey = new ArrayList<>();
    }

    public void setNumTeeth(int teeth) { numTeeth = teeth; }
    public void addPreyAnimal(Animal animal){ prey.add(animal); }
    public boolean isPreyAnimal(Animal animal) { return prey.contains(animal); }
    public String eats() { return "animals" ;}
}

public class Wolf extends Carnivore
{
    private int socialRank;

    public Wolf(String name, int teeth, int rank)
    {
          super(name, teeth);
          socialRank = rank;
    }

    public void rankUp(){ socialRank++; }
    public void rankDown(){ socialRank--; }
}
```

C. Problem Solving

23. Implement the four classes described below using the principals of object-oriented design (information hiding, encapsulation, inheritance, association, code re-use and exception handling).

a. A BankAccount has an account number, a balance and an annual interest rate. It has methods to deposit and withdraw funds (the amounts deposited or withdrawn must be positive) and methods to get the account number, balance and interest rate. It also has a constructor that allows you to specify the account number and annual interest rate. The initial balance of an account is always $0.0. Implement this class below. [8]

```
public class BankAccount
{
    private int number;
    private double balance;
    private double interest;

    public BankAccount(int account, double rate)
    {
          number = account;
          interest = rate;
          balance = 0.0;    // not technically needed
    }

    public void deposit(double amount)
    {
          if(amount < 0)
                throw new IllegalArgumentException("no negatives");
          balance += amount;
    }

    public void withdraw(double amount)
    {
          if(amount < 0)
                throw new IllegalArgumentException("no negatives");
          balance -= amount;
    }

    public int getAccount() { return account; }
    public double getBalance() { return balance; }
    public double getRate() { return interest; }
}
```

b. A CheckingAccount is a BankAccount that has an overdraft limit that can be set when the account is created and can afterwards be accessed but not changed. (An overdraft limit is a number that represents how far in debt the account can go. So if the overdraft limit is 500, the balance on the account can never go lower than -$500.00.) Implement this class below. [10]

```
public class CheckingAccount extends BankAccount
{
    private final double OVERDRAFT;

    public CheckingAccount(int account, double rate, double overdraft)
    {
          super(account, rate);
          OVERDRAFT = overdraft;
    }

    public double getOverdraft() { return OVERDRAFT; }

    @Override
    public void withdraw(double amount)
    {
          if(balance - amount + OVERDRAFT < 0)
                throw new IllegalArgumentException("overdraft exceeded");
    super.withdraw(amount);
    }
}
```

c. A Customer has a name and a BankAccount. It has a constructor that can be used to specify both fields and has methods to get and set the BankAccount. Implement this class below. [5]

```
public class Customer
{
    private String name;
    private BankAccount account;

    public Customer(String name, BankAccount account)
    {
            this.name = name;
        setBankAccount(account);
    }

    public BankAccount getBankAccount() { return account; }
    public void setBankAccount(BankAccount account) { this.account = account; }
}
```

24. Consider the main method below. Depending on how the user behaves, there are 3 exceptions that might be thrown when this code is run. Your job is to figure out which three exceptions might be thrown (see the list of possible exceptions below), then re-write the code with additions to it so that if an exception happens, the program prints a user-friendly message and then starts the program again from the beginning. If the user makes it through with no exceptions, the program should end (i.e. not loop back).

    To get full marks, your re-written code must include all the lines from the original code and should execute them in the original order as long as there are no exceptions thrown. [14]

```
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    int[] x = {1,-1,0,2,-2};
    int a = sc.nextInt();
    int b = sc.nextInt();
    int c = x[a];
    int d = x[b];
    int e = c / d;
    System.out.println(e);
}
```

List of Exceptions

ArithmethmeticException
ArrayIndexOutOfBoundsException
FileNotFoundException
InputMismatchException
InterruptedException
NoSuchElementException

```
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    int[] x = {1,-1,0,2,-2};
    boolean notOK = true;

    while(notOK)
    {
          try
          {
                int a = sc.nextInt();
                int b = sc.nextInt();
                int c = x[a];
                int d = x[b];
                int e = c / d;
                notOK = false;
          }
          catch(ArithmeticException ex)
          {
                System.out.println("Can't divide by 0. Start again.");
          }
          catch(ArrayIndexOutOfBoundsException ex)
          {
                System.out.println("Must be in the range 0 to 4. Start again.");
          }
          catch(InputMismatchException ex)
          {
                System.out.println("Must be an integer. Start again.");
          }
    }

    System.out.println(e);
}
```

25. The code below worked just fine before I added the constructor to it. Now I'm getting a syntax error on the line that's underlined below. NetBeans says, "constructor NewClass in class NewClass cannot be applied to given types;".

```
public class NewClass
{
       public NewClass(int x) { }

       public static void main(String[] args)
       {
              NewClass n = new NewClass();

              System.out.println(n);
       }
}
```

a. Why did I not see this error before I added the constructor? Why am I seeing this error now? [4]

If you don't specify a constructor, Java will provide a default, no-arg constructor. Since the main is calling a no-arg constructor, the code worked when you had not specified a constructor with arguments.

b. I know that I could fix this by removing the constructor. But I don't want to remove the constructor. Give me two other, different ways I could fix this code. [2]

1. Add you own no-arg constructor.
2. Change the main method to use your new 1-arg constructor instead.

26. Consider the code below: [4]

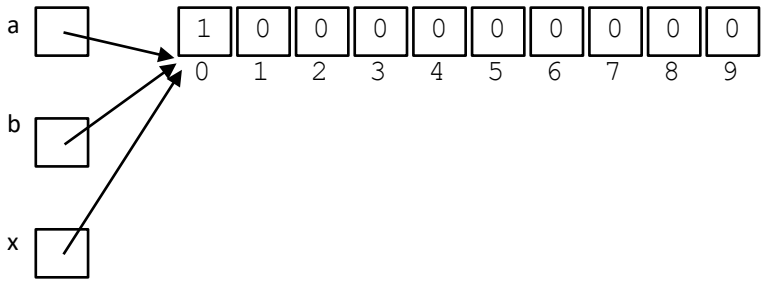
```
public int[] foo(int[] x)
{
      x[0] = x[0] + 1;
      return x;
}

int[] a = new int[10];
int[] b = foo(a);
System.out.println(a[0]+" "+b[0]).
```

a. What will this code output to System.out? [1]

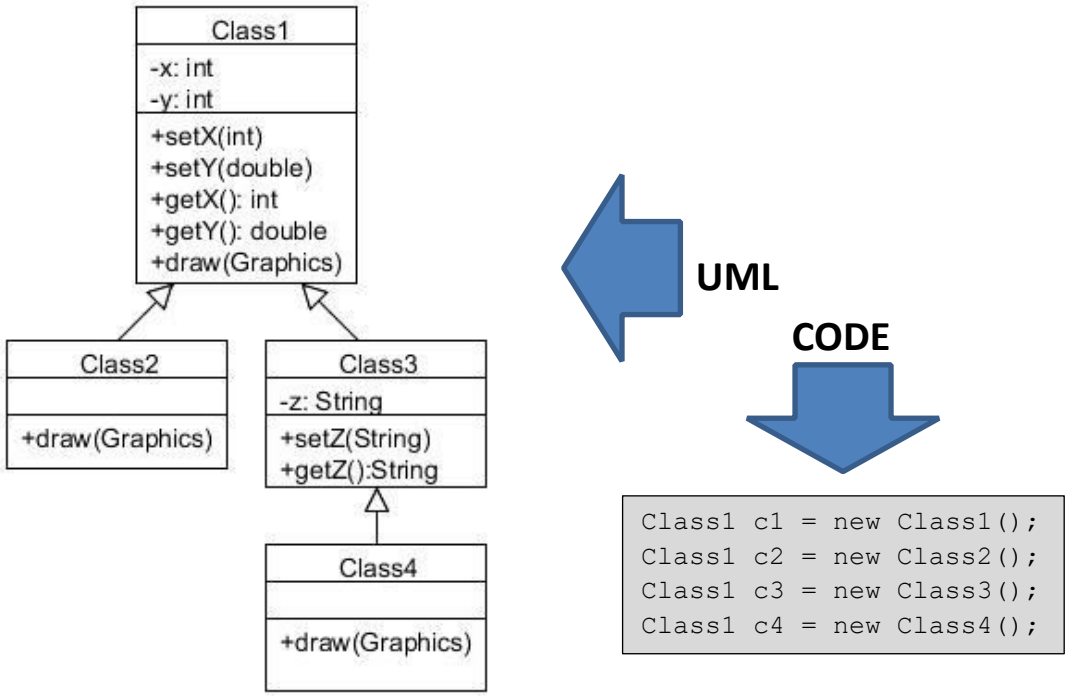
1 1

b. Explain your answer using a picture of the contents of the variables a and b. [3]

All three variables (a, b, and x) reference the same array.



27. Consider the UML Diagram and accompanying code shown below.[6]



a. For each of the following lines of code, indicate by circling whether or not it is legal. [3]

1. `c1.setz("abc.");` Legal Not Legal
2. `c2.setz("abc.");` Legal Not Legal
3. `c4.setz("abc.");` Legal Not Legal
4. `c1.draw(g);` Legal Not Legal Calls Class1's method
5. `c3.draw(g);` Legal Not Legal Calls Class1's method
6. `c4.draw(g);` Legal Not Legal Calls Class4's method

b. For each legal method call in part a, indicate on the UML diagram which method will get executed. (You can show this by writing the appropriate numbers from 1 to 6 on the diagram.) [3]

Shown above.

28. Consider the code below. [4]

```
public class Foobar
{
	private int x;

	public Foobar(int initialX)
	{
		x = initialX;
	}
}

public class Foobar2 extends Foobar
{
	public Foobar2(int initialX){ }
}
```

a. NetBeans is putting a red underline on the constructor for Foobar2. It says there's an error here. Explain why. [2]

Every constructor calls a super constructor. If you don't include an explicit `super()` call, it will call the no-argument super constructor by default. Foobar does not have a no-arg constructor.

b. Describe or demonstrate one way to fix the code so I don't get this error. [1]

```
public Foobar2(int initial) { super(initial); }
```

c. Describe or demonstrate a different way to fix the code so I don't get this error. [1]

Add an overloaded Foobar no-arg constructor.

Or, remove the Foobar constructor (this will cause Java to automatically create a no-arg constructor).

29. Consider the two implementations of the class MyClass below.

```
public class MyClass
{
   public int x;
   public String z
}

public class MyClass
{
   private int x;
   private String z;
   public int getX(){return x;}
   public String getZ() {return z;}
}
```

a. Are these two implementations equivalent from a functional point of view? (I.e. is there anything important that you can do with one that you can't do with the other?) Explain your answer referring to specific features of the code above. [2]

No, the first implementation has public instance variables. This means that other classes can alter the contents of the variables. The second has private instance variables with get methods for each. In this case, the values stored in the variables can be retrieved but not changed.

b. Which implementation represents a better object-oriented design? Give a good reason for your choice. [2]

The second implementation is better because we can have more control over the state of a MyClass object. Private instance variables can only be changed if our code allows it. This means that we can check error conditions before setting the variables. This does not break encapsulation.

30. The code below has some duplication in it. Modify the code or rewrite it to eliminate as much of this duplication as you can. [10]

```
public class AClass
{
	private int x, y, z;

	public AClass(int x, int y, int z)
	{
		this.x = x;
		this.y = y;
		this.z = z;
	}

	public int compute() { return x + y }
	public int calculate() { return x * y; }
}


public class BClass
{
	private int x, y;
	private double a;

	public BClass(int x, int y, double a)
	{
		this.x = x;
		this.y = y;
		this.a = a;
	}

	public int compute() { return x + y; }
}
```

```
public class BaseClass
{
    private int x, y;

    public BaseClass(int x, int y)
    {
          this.x = x;
          this.y = y;
    }

    public int compute() { return x + y; }
}


public class AClass extends BaseClass
{
    private int z;

    public AClass(int x, int y, int z)
    {
          super(x, y);
          this.z = z;
    }

    public int calculate() { return x * y; }
}


public class BClass extends BaseClass
{
    private double a;

    public BClass(int x, int y, double a)
    {
          super(x, y);
          this.a = a;
    }
}
```