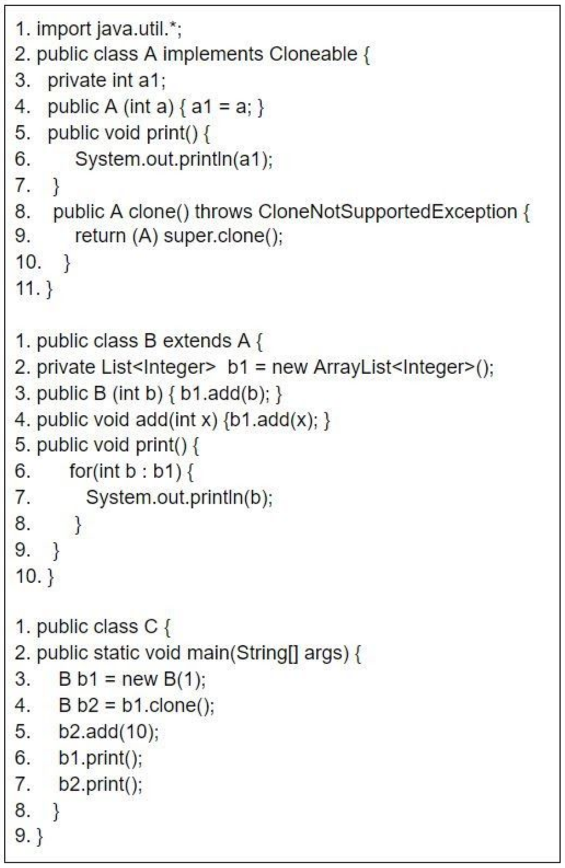
Advanced Programming (CSE201), Endsem Quiz
Time allocated: 10:30am – 12:00noon (1.30 hours), Total Marks: 30
(No Submissions are allowed beyond 12.00noon)

Instructions:

- This is a closed book quiz.
- Only reasonable and clearly mentioned assumptions (if any) would be accepted.
- For justifications, please be as concise as possible (2-3 sentences only)
- **IIIT plagiarism policy is applicable if any such cases found**
- **Write your answers on a plain sheet that you can upload by taking a picture of the same (ensure low resolution so that the upload size is smaller)**
- **You can either submit your solutions on the below Google Form link https://docs.google.com/forms/d/e/1FAIpQLSc3vCUr1_sllYetSf78XuX2p0jMvn9sUa2FhA8yomIpbJ5mew/viewform?usp=sf_link or email the quiz solutions to "ap-m2020-submission@iiitd.ac.in" or upload on google classroom . Only one mode of submission!**
- **Subject of the mail should be the Endsem Quiz.**
- **We will not consider any submission that is submitted beyond 12.00noon. It is your responsibility to ensure you email it or submit through google form or through google classroom on time. Please ensure you have proper internet connectivity as we are giving you sufficient extra time to send the email or upload your solutions either on google classroom or on google form.**

Question-1: Is / Are there any issue in this code (compilation / runtime exception / improper design)? If yes, then identify it and write the correct code. **[7 marks]**

.



```
import java.util.*;
public class A implements Cloneable {
  private int a1;
  public A (int a) { a1 = a; }
  public void print() {
      System.out.println(a1);
  }
   public A clone() throws CloneNotSupportedException {
      return (A) super.clone();
   }
}

public class B extends A {
private List<Integer>  b1 = new ArrayList<Integer>();
public B (int b) { b1.add(b); }
public void add(int x) {b1.add(x); }
public void print() {
     for(int b : b1) {
        System.out.println(b);
     }
  }
}

public class C {
public static void main(String[] args) {
    B b1 = new B(1);
    B b2 = b1.clone();
    b2.add(10);
    b1.print();
    b2.print();
  }
}
```

Solution:
Missing Clone (+1 marks)
Reason: Implementation of clone in B with deep copy of list (+3 marks)

Missing call to super constructor (+1 marks) at Line 3 of Class B
Missing "throws CloneNotSupportedException" (+1 marks) at Line 2 of Class C
Missing mandatory typecasting (+1 marks) at Line 4 of Class C

Question-2: Convertible extends Car and Porsche extends Convertible. Car class has a method **drive** that is overridden in each child class mentioned above.The method **drive_car** below drives different types of Cars. Rewrite the declaration of this static method after performing a type erasure. **[2 marks]**

```
public static <T extends Car> void drive_car(T c) { c.drive(); }
```

Solution: public static void drive_car(Car c) {... } (+2 marks)

Question-3:

a) In the following code, identify the design patterns that have been implemented or used. In each case, explain how you identified the design pattern. **[10 marks]**

b) Draw the UML diagram based on the code. **[5 marks]**

```
abstract class Noname0 {
        public final void noMethodName0_0() {
                noMethodName0_1();
                noMethodName0_2();
                noMethodName0_3();
                noMethodName0_4();
        }
        public void noMethodName0_1() {
                System.out.println("This is noMethodName0_1 in Noname0");
        }
        public void noMethodName0_3() {
                System.out.println("This is noMethodName0_3 in Noname0");
        }
        public abstract void noMethodName0_2();
        public abstract void noMethodName0_4();
}
```

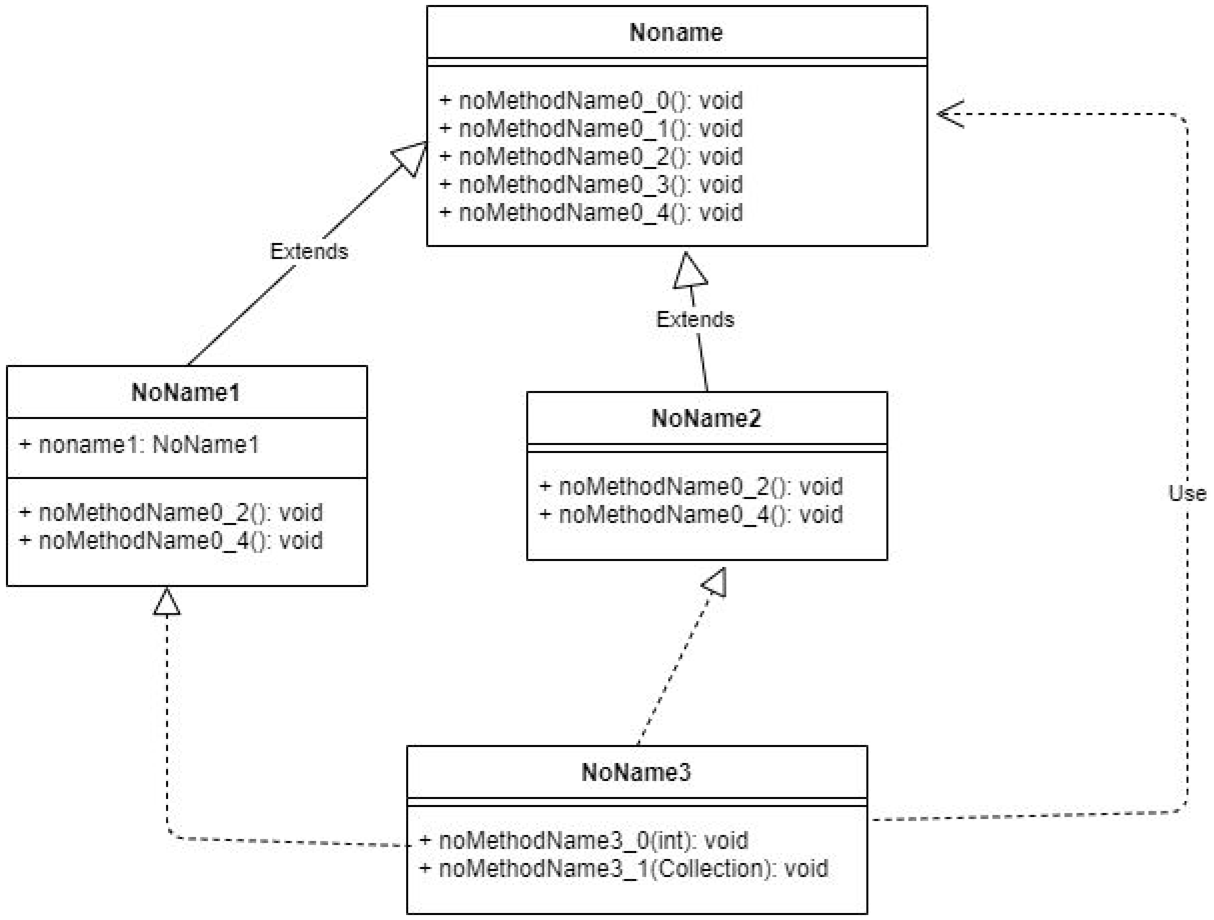
```java
class Noname1 extends Noname0 {
    private static Noname1 noname1;
    private Noname1() { }
    public static synchronized Noname1 noMethodName1_0() {
        if(noname1 == null)
            noname1 = new Noname1();
        return noname1;
    }
    @Override
    public void noMethodName0_2() {
        String s1 = "abc";
        String s2 = new String("abc");
        System.out.println(s1 + "and " + s2 + " are equal");
        System.out.println("This is noMethodName0_2 in Noname1");
    }
    @Override
    public void noMethodName0_4() {
        System.out.println("This is noMethodName0_4 in Noname1");
    }
}
```

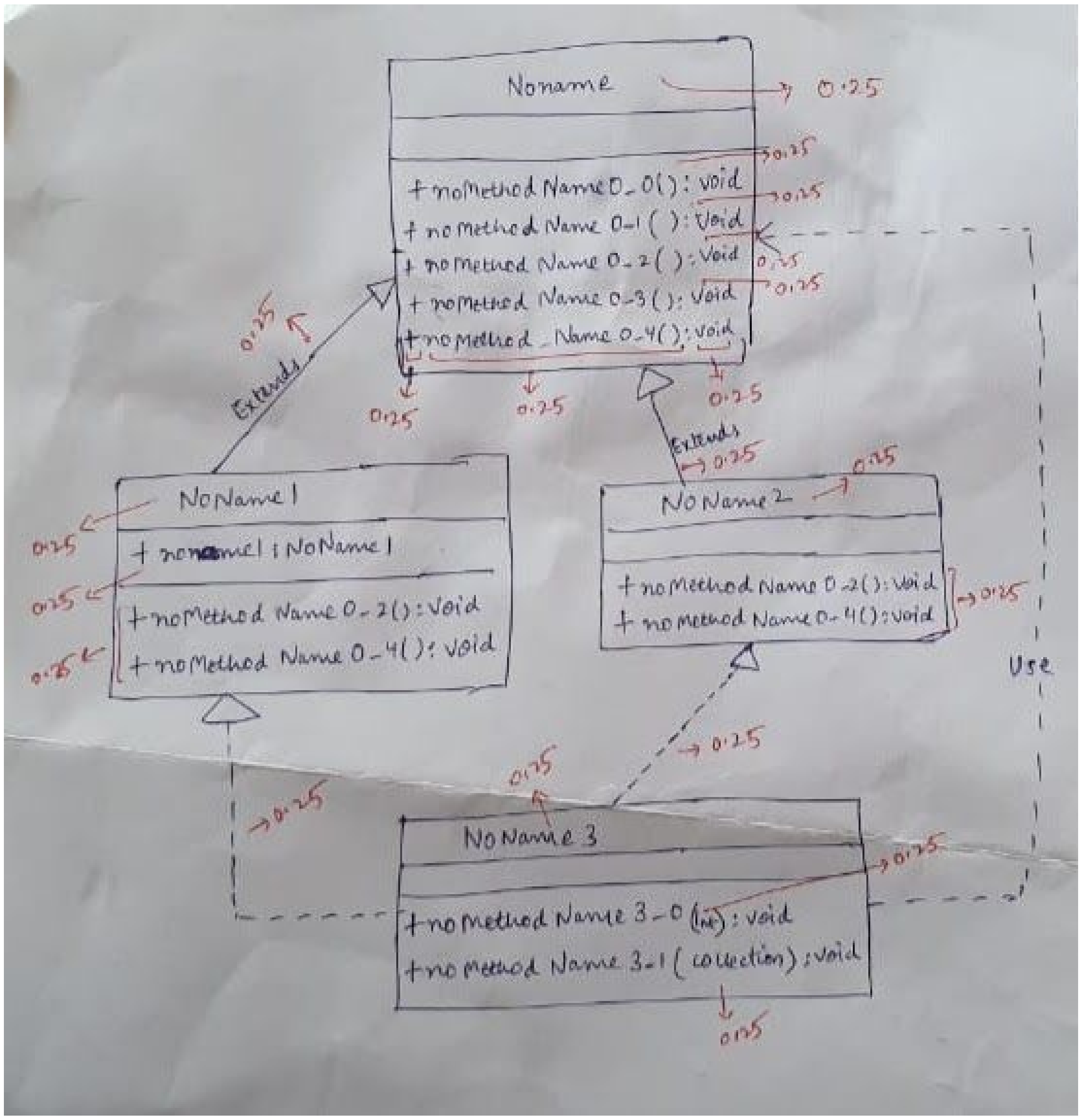
```java
class Noname2 extends Noname0 {
    @Override
    public void noMethodName0_2() {
        System.out.println("This is noMethodName0_2 in Noname2");
    }
    @Override
    public void noMethodName0_4() {
        System.out.println("This is noMethodName0_4 in Noname2");
    }
}

class Noname3 {
    public Noname0 noMethodName3_0(int option) {
        if(option == 1)
            return Noname1.noMethodName1_0();
        else if(option == 2)
            return new Noname2();
        else
            return null;
    }
    public void noMethodName3_1(Collection<Noname2> collection) {
        Iterator<Noname2> collt = collection.iterator();
        while(collt.hasNext()) {
            System.out.println(collt.next());
        }
    }
}
```

Solution: There are 5 design patterns used - Singleton, Factory, Template, Iterator, and Flyweight. (1*5 =5 marks)

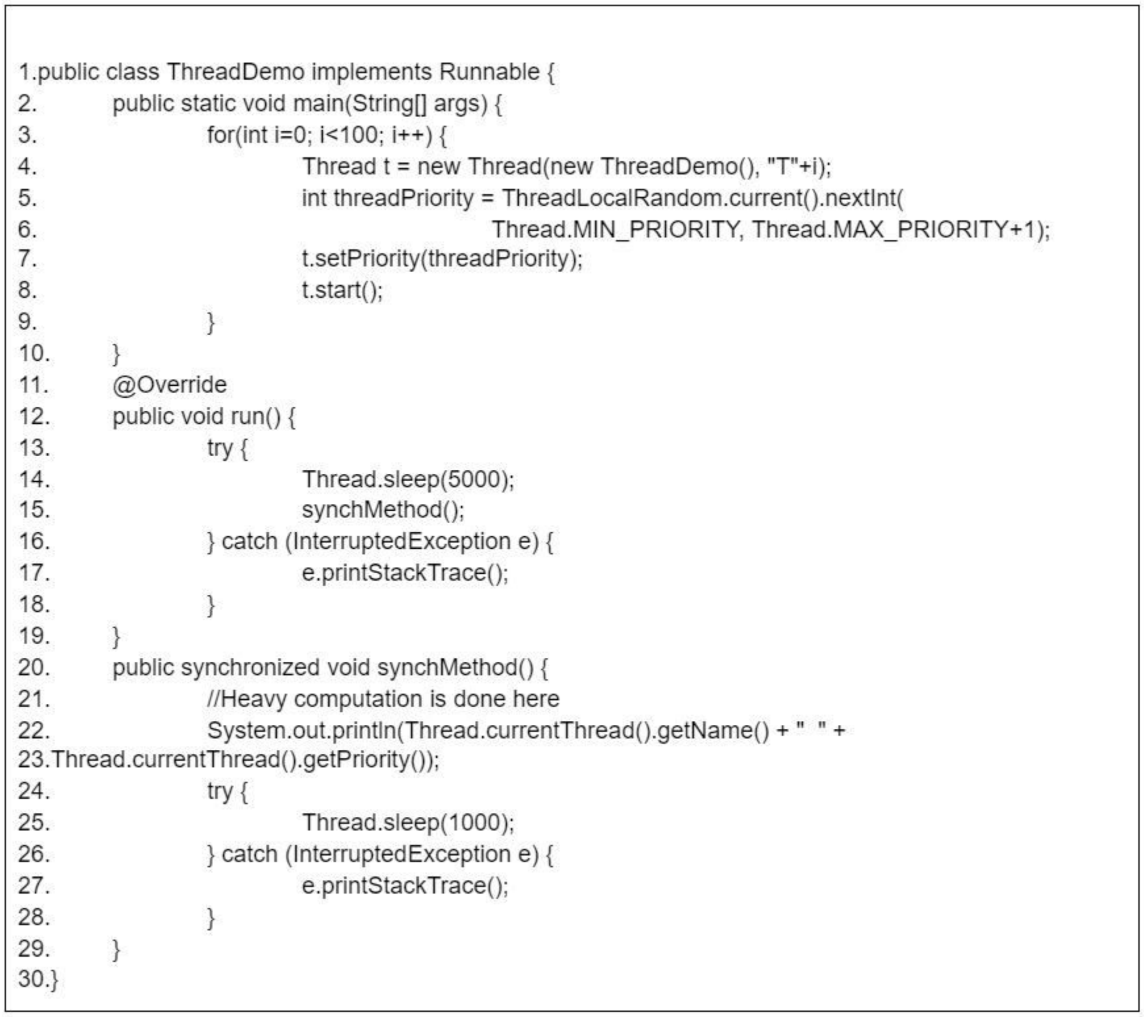
1. Singleton : It is seen in class Noname1 where only one instance of type Noname1is created and its constructor has also been made private. The static method noMethodName1_0 has been used and it creates a new instance only if the static field nonname1 is null otherwise it returns the same instance. **(+1 mark)**
2. Template : Class Nonname0 is abstract and it has two classes inheriting from it and overrides two functions (abstract methods) but contains the same definition of the two non abstract methods. **(+1 mark)**
3. Factory : In class Nonname3, the method noMethodName3_0 is responsible for creation of objects of inherited classes of Noname0 **(+1 mark)**
4. Iterator : The method noMethodName3_1 in class Noname3 iterating through collections and the type of iterator is generic. **(+1 mark)**
5. Flyweight : The function noMethodName0_2() is flyweighted due to the implementation of string in JVM. **(+1 mark)**

b) UML diagram


Noname
+ noMethodName0_0(): void
+ noMethodName0_1(): void
+ noMethodName0_2(): void
+ noMethodName0_3(): void
+ noMethodName0_4(): void
Extends
Extends
NoName1
+ noname1: NoName1
+ noMethodName0_2(): void
+ noMethodName0_4(): void
NoName2
+ noMethodName0_2(): void
+ noMethodName0_4(): void
Use
NoName3
+ noMethodName3_0(int): void
+ noMethodName3_1(Collection): void


Noname
0.25
+ noMethod Name 0_0( ): void
+ no Method Name 0-1 ( ): Void
+ no Method Name 0_2 ( ): Void
+ noMethod Name 0-3 ( ): Void
+ noMethod_Name 0_4( ): void
Extends
NoName1
+ nonamel : NoName1
+ noMethod Name 0_2( ): Void
+ noMethod Name 0_4( ): void
NoName2
+ noMethod Name 0_2( ): void
+ no Method Name 0-4( ): void
Use
NoName 3
+ no Method Name 3_0 (int): void
+ no Method Name 3-1 (collection) : void

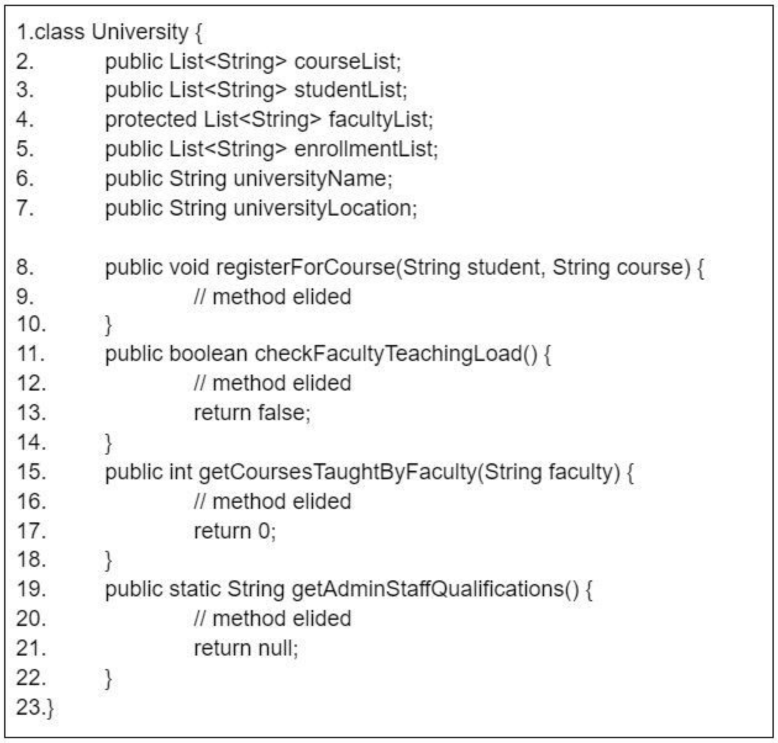
Question-4: Would there be any issues with the following code (assume there are no compilation or runtime errors). What could be the reason(s) for those issues and how would you resolve them? **[3 marks]**



```
public class ThreadDemo implements Runnable {
        public static void main(String[] args) {
                for(int i=0; i<100; i++) {
                        Thread t = new Thread(new ThreadDemo(), "T"+i);
                        int threadPriority = ThreadLocalRandom.current().nextInt(
                                        Thread.MIN_PRIORITY, Thread.MAX_PRIORITY+1);
                        t.setPriority(threadPriority);
                        t.start();
                }
        }
        @Override
        public void run() {
                try {
                        Thread.sleep(5000);
                        synchMethod();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
        public synchronized void synchMethod() {
                //Heavy computation is done here
                System.out.println(Thread.currentThread().getName() + "  " +
Thread.currentThread().getPriority());
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

Solution:
This code could lead to starvation among threads (+0.5 marks).
There could be two reasons for it:
1) Higher priority threads get scheduled before lower priority threads. (+1 marks)
2) Some threads could keep waiting for the lock due to the presence of the synchronized method. (+1 marks)
In order to avoid the starvation, assign the same priority to all the threads. (+0.5 marks)
Additional answer which is optional -- use locking mechanisms that take fairness into account.

Question-5: Identify the issues with the following code in terms of bad object-oriented design choices. Why are they considered bad? **[3 marks]**



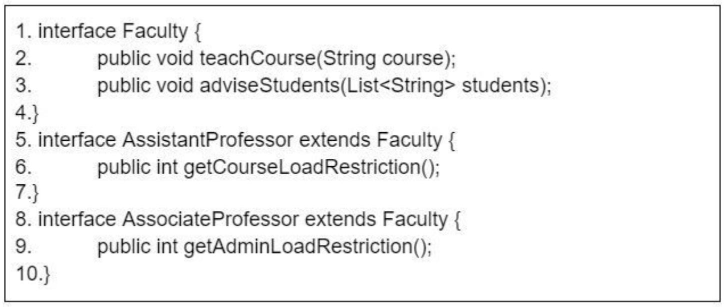
```
class University {
        public List<String> courseList;
        public List<String> studentList;
        protected List<String> facultyList;
        public List<String> enrollmentList;
        public String universityName;
        public String universityLocation;

        public void registerForCourse(String student, String course) {
                // method elided
        }
        public boolean checkFacultyTeachingLoad() {
                // method elided
                return false;
        }
        public int getCoursesTaughtByFaculty(String faculty) {
                // method elided
                return 0;
        }
        public static String getAdminStaffQualifications() {
                // method elided
                return null;
        }
}
```

Solution:

Use of constructor (+0.5 marks)

Appropriate getters and setters to access data. (+0.5 marks)

Modifiers (public, static) are not appropriate. Encapsulation is broken. (+1 marks)



```
interface Faculty {
        public void teachCourse(String course);
        public void adviseStudents(List<String> students);
}
interface AssistantProfessor extends Faculty {
        public int getCourseLoadRestriction();
}
interface AssociateProfessor extends Faculty {
        public int getAdminLoadRestriction();
}
```

Solution:

Faculty should be an abstract class. AssistantProfessor and AssociateProfessor should be classes that inherit from the abstract class. Some of the common functionality/implementation can be provided by the abstract class, but not by an interface. (+1 marks)