Module 9 Object Orientated Programming

Module title	Object Orientated Programming
Module NFQ level (only if an NFQ level can be demonstrated)	6
Module number/reference	BSCH-OOP
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Stage 2
Semester (semester1/semester2 if applicable)	Semester 1
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	10
List the teaching and learning modes	Direct, Blended
Entry requirements (statement of knowledge, skill and competence)	Learners must have achieved programme entry requirements.
Pre-requisite module titles	BSCH-CP
Co-requisite module titles	None
Is this a capstone module? (Yes or No)	No
Specification of the qualifications (academic, pedagogical and professional/occupational) and experience required of staff (staff includes workplace personnel who are responsible for learners such as apprentices, trainees and learners in clinical placements)	Qualified to as least a Bachelor of Science (Honours) level in Computer Science or equivalent and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.
Maximum number of learners per centre (or instance of the module)	60
Duration of the module	One Academic Semester, 12 weeks teaching
Average (over the duration of the module) of the contact hours per week	5
Module-specific physical resources and support required per centre (or instance of the module)	One class room with capacity for 60 learners along with one computer lab with capacity for 25 learners for each group of 25 learners

Analysis of required learning effort		
	Minimum ratio teacher / learner	**Hours**
Effort while in contact with staff		
Classroom and demonstrations	1:60	24
Monitoring and small-group teaching	1:25	36
Other (specify)		
Independent Learning		
Directed e-learning		
Independent Learning		100
Other hours (worksheets and assignments)		90
Work-based learning – learning effort		
Total Effort		250

Allocation of marks (within the module)					
	Continuous assessment	**Supervised project**	**Proctored practical examination**	**Proctored written examination**	**Total**
Percentage contribution	60%			40%	100%

Module aims and objectives

This module builds on the work completed in the first year Computer Programming module and extends the learners knowledge of programming by giving a comprehensive analysis of object-oriented programming. This paradigm leads to software architectures based on the objects every system or subsystem manipulates. In this view software systems are operational models of real or virtual world activities based around the objects that populate these worlds: people, cars, houses, stacks, sets, queues. As in all programming modules, a key objective is the acquisition, on behalf of the learner, of good software engineering skills and the application of these skills to the design and implementation of software components.

Minimum intended module learning outcomes

On successful completion of this module, the learner will be able to:

1. Explain the main reasons behind the development of the object-oriented model of software development
2. Implement classes that encapsulate both simple and complex behaviours
3. Explain the relationship between encapsulation and public interfaces
4. Define both inheritance and composition and the differences between them
5. Design and implement classes that use inheritance and composition
6. Apply abstract concepts in an object-oriented manner
7. Develop high quality software that is reliable, reusable and maintainable

Rationale for inclusion of the module in the programme and its contribution to the overall MIPLOs

Computer programming is a fundamental skill in computing science. This module expands the domain of knowledge for the Learner and introduces a new programming paradigm. Appendix 1 of the programme document maps MIPLOs to the modules through which they are delivered.

Information provided to learners about the module

Learners receive a programme handbook to include module descriptor, module learning outcomes (MIMLO), class plan, assignment briefs, assessment strategy and reading materials.

Module content, organisation and structure

Introduction and motivation

- Review of procedural paradigm and its limitations.
- Outline of key reasons for development of object-oriented paradigm

Classes and Objects

- Encapsulation: class definition, private, public modifiers, public methods.
- Examples of class definitions and programs that interact with public class interfaces.

Composition and Inheritance

- Composing new classes from existing classes.
- Protecting encapsulation. Inheritance and class hierarchies.
- Access modifier protected.
- Polymorphism, multiple inheritance and interfaces.
- Abstract classes.

Immutable objects

- Defining classes that have immutable state.
- Examples from Java – String, Integer, Double, Boolean, Character.

Object class

- Need to override methods toString, equals and hashCode.
- The comparable interface and implementing the compareTo method.
- Examples of classes that implement comparable interface and override equals, hashCode and toString. Hashing functions.

- Issues around problem of cloning and copy constructors.
- The equals contract in Java and issues that arise around inheritance and satisfying the equals contract.

Static Members and Enumerated Types

- Class static members.
- Singleton classes.
- Enumerated types.
- Programming examples of each one.

Robustness and Exceptions

- Examples of the different types of exception and methods for both throwing and catching exceptions.
- Programming with exceptions.
- Writing exception handlers.

Collection classes

- Genericity and the Collection classes.
- Sets, Lists, ArrayLists and Maps.
- Using collection classes and user-defined classes.
- Traversing collections.

Module teaching and learning (including formative assessment) strategy

The module is delivered through a combination of lectures and practical lab programming sessions. The learners complete a series of worksheets throughout the module that are directly related to the material covered in lectures. The emphasis is on developing sound software engineering skills in practical programming based on theoretical knowledge.

Assessment consists of a series of continuous assignments and a final examination. Each week learners are required to complete a series of programming tasks that relate to the material covered in lectures. The practical lab sessions are used to enforce concepts covered in the lectures and the worksheets are used to ensure that learners are keeping up with the material as it is delivered. Lab sessions are also used to deal with issues emerging from the worksheets. All work submitted by learners is assessed and comments are given to individual learners. Typically, there are 10 worksheets and the final mark is based on the 7 best pieces of work submitted.

Timetabling, learner effort and credit

The module is timetabled as one 2-hour lecture and two 1.5-hour labs per week.

The number of 10 ECTS credits assigned to this module is our assessment of the amount of learner effort required. Continuous assessment spreads the learner effort to focus on small steps before integrating all steps into the overall process of computer program design and implementation.

There are 60 contact hours made up of 12 lectures delivered over 12 weeks with classes taking place in a classroom. There are also 24 lab sessions delivered over 12 weeks taking place in a fully equipped computer lab. The learner will need 100 hours of independent effort to further develop the skills and knowledge gained through the contact hours. An additional 90 hours are set aside for learners to work on worksheets and assignments that must be completed for the module.

The team believes that 250 hours of learner effort are required by learners to achieve the MIMLOs and justify the award of 10 ECTS credits at this stage of the programme.

Work-based learning and practice-placement

There is no work based learning or practice placement involved in the module.

E-learning

The college VLE is used to disseminate notes, advice, and online resources to support the learners. The learners are also given access to Lynda.com as a resource for reference.

Module physical resource requirements

Requirements are for a classroom for 60 learners equipped with a projector, and a 25-seater computer lab for practical sessions with access to Java and a suitable development environment (for example Notepad++) (this may change should better technologies arise).

Reading lists and other information resources

Recommended Text

Deitel, P. J. and Deitel, H. M. (2018) *Java: How to Program*. New York: Pearson.

Secondary Reading:

Baesens, B. and Vohra, D. (2015) *Beginning Java Programming: the Object Oriented Approach*. Hoboken: Wiley.

Nino, J. and Hosch, F. A. (2008) *Introduction to Programming and Object Oriented Design using Java*. Hoboken: Wiley.

Meyer, B. (1997) *Object-oriented Software Construction*. Upper Saddle River, NJ: Prentice Hall.

Specifications for module staffing requirements

For each instance of the module, one lecturer qualified to at least Bachelor of Science (Honours) in Computer Science or equivalent, and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.. Industry experience would be a benefit but is not a requirement.

Learners also benefit from the support of the programme director, programme administrator, learner representative and the Student Union and Counselling Service.

Module Assessment Strategy

The assignments constitute the overall grade achieved, and are based on each individual learner's work. The continuous assessments provide for ongoing feedback to the learner and relates to the module curriculum.

No.	Description	MIMLOs	Weighting
1	A series of worksheets: Each will relate to the current topic covered in lectures. Cumulatively they will enforce learning outcomes 1-8	1-7	60%
2	Written exam that tests the theoretical aspects of the module	1-7	40%

All repeat work is capped at 40%.

Sample assessment materials

Note: All assignment briefs are subject to change in order to maintain current content.

Worksheet 1

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload a copy of your solution on Moodle using the template program listed on Moodle. Please remember to include your name and student number.

Question 1

Implement a class to model an *on/off* switch. Your class should have methods to turn on the switch, turn off the switch and check its current status. The class should have a **toString** method that returns its current state as a String. Write a short code block in to test your class in the file given for this assignment.

Question 2

Design and implement a class called *Film* that has the following attributes: name of film, name of director, duration of film measured in minutes and cost of production. The class should have methods to retrieve information on the current state of each of its attributes and a **toString** method that can be used to print its current state. In the file given for the assignment create 3 instances of the Film class and write a code fragment that finds the film with the most expensive production cost.

Worksheet 2

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload a copy of your solution on Moodle using the template program listed on Moodle. Please remember to include your name and student number.

Question 1

A circle in co-ordinate Geometry has a centre, represented by a point in the plane, and a radius. Write a class that represents a circle. Your class should have methods that:

- return the centre of the circle;
- returns its radius
- calculates its area
- calculates its circumference
- calculates the distance of its centre point from that of a given circle c1
- determines if a point is on the circle
- determines if it intersects with a given circle c1

Question 2

Write a class Email that has two string attributes: home that records the home e-mail address and work that records the work e-mail. It should have methods to retrieve

both addresses and also methods that change email addresses. You may assume that both e-mail addresses are valid.

Write a class Employee that has three attributes: a surname, a first name and an e-mail address represented by the class Email. Your constructor should take 3 arguments: surname, firstName and an instance of the Email class.

This class should have methods that return: the name as a string representing both surname and first name and the e-mails of the employee.

The important point is to write this Employee class so that **encapsulation** is not endangered by any of its public methods

Worksheet 3

Please complete the problems listed below. This assignment forms part of the assessment for this module and you are required to upload your solution to Moodle before mid-night on Sunday next. A template program is available on Moodle. You must include as header your name, student number and the assignment number.

Question 1

The class IntManager listed below manages an array of Integer values. Your task is to complete the methods listed in the interface.

```
class IntManager{
        private Integer dt[];
        private int size;
        IntManager(int k){dt = new Integer[k]; size = 0;}
        public void add(Integer x){
                if(size < dt.length){
                        dt[size] = x; size++;
                }
        }
        /*public boolean found(Integer x){
                //return true if x in dt; false otherwise
        }
        public Integer max(){
          //return largest value in dt; null if size == 0
        }
        public Integer sumOdd(){
                //calculate sum of odd values
        }
        public Integer freq(Integer x){
                //count frequency of occurrence of x in dt
        }
```

```
   public void sort(){}
   public Integer[] getOdd(){} */
        public String toString(){
                if(size == 0) return "[]";
                String s = "[";
                for(int j = 0; j < size - 1; j++)
                        s = s + dt[j] + ",";
                return s+dt[size-1]+"]";
        }
}
```

Question 2

A local lottery sells a small number of tickets for a draw. Each ticket has only two numbers selected at random. Numbers are restricted to values in the range 0.5. The class Ticket that encapsulates an individual ticket is given. A class TicketManager is also given and its methods are listed in the table below. Your task is to complete the methods listed. The toString method is given and should not be modified by you.

Method	Semantics
TicketManager()	Constructor that creates an array of size maxTickets.
public boolean buy(Ticket t)	Adds a ticket to the draw on condition that the number sold does not exceed maxTickets. It returns true if successful, false otherwise.
public int freqWinner(Ticket t)	It checks for the number of winning tickets after a draw takes place.
public boolean search(Ticket t)	Searches for the given ticket and returns true if found; false otherwise.
public int sold()	Returns the number of tickets sold.
public boolean allsold()	Returns true if all tickets sold; false otherwise.

Worksheet 4

Please complete the question listed below. This assignment forms part of the assessment for this module and you are required to upload a copy of your work on Moodle on or before next Sunday before midnight. You must include as header your name, student number and the assignment number.

Question 1

(a) A computer program for a certain university has to manipulate both lecturers and students. A class is required for each. Write a class Person that encapsulates what is common to the notion of a person. The class should include a name, a phone number, a gender and an all argument constructor. Methods that retrieve name, phone and gender should be included. It should also have a toString() method that returns a string representing the state of the class.

(b) A student is a person with additionally a student number and a subject of study. Write a class Student by extending class Person. It should include a toString() method and methods to retrieve name, phone, gender, student number and subject.

(c) A lecturer is a person with additionally a staff number and a department. Write a class Lecturer by extending class Person. It should include a toString() method and methods to retrieve details about a lecturer.

(d) Write a simple test program that tests your classes.

Question 2

Write a class Lecturers that encompasses a collection of Lecturer. It should be possible to add a new lecturer, retrieve all lecturers of a given gender, retrieve all lecturers working in a given department and find a lecturer given their staff number.

Worksheet 5

Question 1

(a) A square is a shape where all sides are the same length. Using the abstract class Shape write a class to encapsulate a square. A square should also have a diagonal method that returns the length of its diagonal.

(b) A circle is a shape defined by its radius. Using the abstract class Shape write a class to encapsulate a circle. In the case of a circle the perimeter is its circumference length.

Question 2

A rational number is any number that can be expressed as the quotient or fraction of two integer values, with the denominator not equal to 0. That is, any number that can be written in the form $\frac{a}{b}, where\ a, b \in \mathbb{Z}\ and\ b \neq 0$. Everyone learned to work with fractions at some stage in their youth so we just present, without comment, a list of operators together with their definitions.

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

$$\frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd}$$

$$\frac{a}{b} * \frac{c}{d} = \frac{ac}{bd}$$

$$\frac{a}{b} \Big/ \frac{c}{d} = \frac{a}{b} * \frac{d}{c} = \frac{ad}{bc}$$

We will use these definitions when we implement operators for our rational numbers.

An interesting observation about rational numbers is that equal values can have different forms. For example, $\frac{1}{2} = \frac{2}{4} = \frac{4}{8} = \frac{8}{16} = \cdots$ The normalized value of all of these fractions is $\frac{1}{2}$ because each of the other values can be simplified to this value.

Note that when you add one rational number to another one the result of the addition is a new rational number. The original values used in the addition are not changed or modified. Therefore, operations add, sub, mult, etc all return new Rational numbers.

```
interface Operations{
    public Rational add(Rational q);
    public Rational sub(Rational q);
```

```
    public Rational mult(Rational q);
    public Rational multBy(int k);
    public Rational div(Rational q);
     public Rational divBy(int k);
    public boolean eq(Rational q); //returns true if this equals q
}
```

The class, as presented below, has a single constructor that takes two integer arguments It assumes that the denominator, d, is not zero. It also has two attributes, num and den, that refer to the *numerator* and *denominator* of the fraction. The private function gcd, greatest common divisor, is used to ensure that all fractions are stored in normalized form. This function calculates the greatest common divisor using only the absolute values of both the numerator and denominator. Both n and d are divisible by g.

```
class Rational implements Operations{
    private int num,den;
    public Rational(int n, int d){//assume d != 0
     if(n < 0 && d < 0){ n = -n; d = -d;}
     else if(d < 0){ n = -n; d = -d;}
     //ensures that d never negative and n positive e.g. 1/-2 is changed to -1/2
     int g = gcd(Math.abs(n), Math.abs(d));
     num = n/g;den = d/g;
    }
    public Rational(int n){//d == 1  ...}
    public int num(){return num;}
    public int den(){return den;}
    public Rational add(Rational q){...}
    public Rational mult(Rational q){...}
    // ... all the other required methods
    public String toString(){...}
    private int gcd(int a, int b){
     if(b == 0) return a;
     else return gcd(b,a%b); }}
```

Worksheet 6

Question 1

Given below is a mutable Car class. Re-write it so that it becomes immutable and write a simple test program.

```
class Car{
    private String owner;
    private String reg;
    private String make;
    private int kilometres;
    private double price;
    public Car(String ow, String r, String m, int k, double p){
     owner = ow; reg = r; make = m; kilometres = k; price = p;
    }
    public String owner(){return owner;}
    public String reg(){return reg;}
    public String make(){return make;}
    public int kilometres(){return kilometres;}
    public double price(){return price;}
    public void setPrice(double p){price = p;}
    public void setOwner(String ow){owner = ow;}
    public void setKil(int k){kilometres = k;}
    public void setMake(String m){make = m;}
}
```

Question 2

Complete the class MyGarage that implements the interface Garage. This interface has 7 methods. The code for the interface is given in the file Assignment6_2017.java

add(Car c) – add new car to the collection of cars ensuring that its registration number is unique. If the registration number already exists do not add it and return false.

getCar(String reg) – search for the car with registration number reg.

getMake(String make) – returns a list of cars that match the given make.

totalValue() – calculates the total value of all cars in the list.

changeOwner(String reg, String ow) – change the owner of car that has registration number reg to ow.

changePrice(String reg, double p) – change price of car given registration number.

reducePricesBy(double per) – reduce prices of all cars by the given percentage.

Worksheet 7

Question 1

Listed below is an immutable class Person that has a prsi number (prsi), a name, and a date of birth. Two persons are said to be if and only if equal is they have the same prsi number. Write the following methods for your Person class: equals, compareTo, toString and hashCode. The comparison for ordering purposes is based solely on the prsi number. Write a simple test that creates two instances of your Person class and tests each of the methods you have written. Your test program should also create a small array of persons and use the method Arrays.sort() to sort the array and then print it. **Note**: to use this method you must import java.util.*.

```
final class Person implements Comparable<Person>{{
    private final String prsi;
    private final String name;
    private final String dob;
    Person(String p, String n, String d){
        prsi = p; name = n; dob = d;
    }
    public String prsi(){return prsi;}
    public String name(){return name;}
    public String dob(){return dob;}
}
```

Question 2

An orthogonal vector is defined by two real values a, b and written in the form $ai + bj$. We define equality for two vectors $v1 = ai + bj$ and $v2 = ci + dj$ as follows:

$$v1 == v2 \equiv a == c \;\&\&\; b == d$$

Given is a class Vector that encapsulates an orthogonal vector. Your task is to write an immutable version of this class and implement the following methods: toString that returns a string representation of the vector; equals that returns true if this equals that, false otherwise; compareTo that implements the Comparable interface and hashCode. The comparison, for compareTo, is based on a comparison of the coefficients of i, and if equal, on the coefficients of j.

```
class Vector implements Comparable<Vector>{
    private double a,b;
    Vector(double a0, double b0){a = a0; b = b0;}
}
```

Worksheet 8

Question 1

Static members are often used to represent data or calculations that do not change in response to object state. To illustrate this write a class that has static methods that convert Fahrenheit to Celsius, Celsius to Fahrenheit, miles to kilometers and kilometers to miles. The formulas in each case are: f = (c * 9.0/5.0) + 32, c = (f – 32)*5.0/9.0, k = 1.609*m, m = k /1.609. Your class should be called Converter. Write a simple program to test your class.

Question 2

A module encapsulates three attributes: title of module, name of lecturer and the number of hours for delivery. An outline listing of the class and its attributes are given in the accompanying file. An equals method, a toString method and a hashCode are given. You are required to write a compareTo method that implements the Comparable<Module> interface and a Comparator that orders modules based on lecturer names. It should be called lectCompare and its signature is given as part of the class. **Note:** *This comparator should be consistent with the equals method given.*

The file *Assignment8_2017.java* contains a short list of module instances. Write code to sort and print the list using both the natural ordering and the lecturer comparator.

Question 3

Write a class called Season that enumerates the seasons in the year. This class should have a toString method that *pretty* prints the names of the season. Now write a class Month that enumerates the months of the year. This class should have a toString method that *pretty* prints the name of the month and a public method season that takes a Month as argument and returns the Season it falls in. Write a test program for your class.

Worksheet 9

Using the class Person defined below create a class Friends that manages a set of Person. This class should have the following methods:

add(Person p) – add new person to set;

search(Person p) – returns true if p is a member of the set of friends, false otherwise;

size() – returns number of friends

getSurname(String s) – returns a set of just those persons whose surname is s;

getFirstname(String f) - returns a set of just those persons whose surname is f;

getFreq(String f) – returns number of persons whose first name is f;
del(Person p) – remove person p if present;
sort() – returns a sorted list of Person.

You may use a TreeSet or a HashSet for your collection of Person.

```
final class Person implements Comparable<Person>{
        private final String sName;
   private final String fName;
        Person(String fn, String sn){fName = fn; sName = sn;}
        public String sName(){return sName;}
   public String fName(){return fName;}
        public String toString(){return fName+" "+sName;}
        public boolean equals(Object ob){
                if(!(ob instanceof Person)) return false;
                Person p = (Person)ob;
                return sName.equals(p.sName) && fName.equals(p.fName);
        }
        public int compareTo(Person p){
                if(p == null) return -1;
   if(this.equals(p)) return 0;
                return sName.compareTo(p.sName);
        }
        public int hashCode(){
     return 41 * sName.hashCode() * fName.hashCode();
   }
}
```

GRIFFITH COLLEGE DUBLIN

QUALITY AND QUALIFICATIONS IRELAND
EXAMINATION

OBJECT ORIENTED PROGRAMMING

Lecturer(s):

External Examiner(s):

Date: **XXXXXXXX** **Time: XXXXXXX**

THIS PAPER CONSISTS OF TWELVE QUESTIONS
TEN QUESTIONS TO BE ATTEMPTED
ALL QUESTIONS CARRY EQUAL MARKS

APPENDIX AT BACK OF EXAMINATION PAPER

QUESTION 1

(a) Implement a class called Phone, whose constructor and public methods are listed in the table below. Each phone has attributes: make, colour and cost. These attributes have types String, String, double, respectively.

Method	Semantics
Phone(String m, String c, double cst)	Constructor method for class
make()	Returns the make of the phone
colour()	Returns the colour of the phone
cost()	Returns the cost of the phone
toString()	Returns a string representation of an instance of a phone

(6 marks)

(b) Write a code fragment that creates two instances of your Phone class and then uses an if statement to determine the more expensive of the two phones.

(4 marks)

Total (10 marks)

QUESTION 2

(a) Explain the difference between primitive variables and reference variables. The code fragment given below initializes variable x to the value 10. Draw a diagram of the state of x after the initialization.

```
Integer x = 10;
```

(4 marks)

(b) Explain why the method player() in the class Player, given below, violates the rule on encapsulation and the method team() does not. Re-write method player() so that encapsulation is preserved.

```
class Person{
    private String name;
    public Person(String n){name = n;}
    public String name(){return name;}
    public void setName(String n){name = n;}
}
class Player{
    private Person plr;
    private String team;
    public Player(String n, String t){plr = new Person(n); team = t;}
    public Person player(){return plr;}
    public String team(){return team;}
}
```

(6 marks)

Total (10 marks)

QUESTION 3

(a) The program Q3, listed below, creates an array of 3 Person instances. Your task is to draw diagrams of the state of the array after execution of the line

```
Person dt[] = {new Person("John",21),
new Person("Nora",19),
new Person("Sheila",22)};
```

and, again, after execution of the code:

```
for(Person p : dt) p.setAge(p.age()+1);
```

(8 marks)

(b) You should also state the output from a single execution of the program.

```
class Q3{
  public static void main(String[] args) {
                    Person dt[] = {new Person("John",21),
                new Person("Nora",19),
                new Person("Sheila",22)};
                for(Person p: dt) System.out.println(p);
                for(Person p : dt) p.setAge(p.age()+1);
                for(Person p: dt) System.out.println(p);
   }
}
class Person{
 private String name;
 private int age;
 public Person(String n, int a){name = n; age = a;}
 public int age(){return age;}
 public void setAge(int x){age = x;}
 public String toString(){return name+" "+age;}
}
```

(2 marks)

Total (10 marks)

QUESTION 4

(a) New classes can be constructed from existing classes by using *composition* or *inheritance*. Explain each of these terms.

(4 marks)

(b) Given is the class TD that has a name and a constituency. Write a class Minister, using either inheritance or composition, that is a TD with a department. You may implement the department name as a String. Class Minister must have a toString method.

```
class TD{
    private String name;
    private String constituency;
    public TD(String n, String c){
            name = n; constituency = c;
    }
```

```
    public String name(){return name;}
    public String constituency(){return constituency;}
    public String toString(){return name+" "+constituency;}
}
```

(6 marks)

Total (10 marks)

QUESTION 5

(a) In relation to the primitive type int and class Integer explain the meaning of the terms *autoBoxing* and *autoUnboxing*.

(3 marks)

(b) What are static methods? Explain the difference between *things that belong to the class itself* and *those that belong to an instance of the class.*

(4 marks)

(c) When you try to compile the class Test below the compiler will generate the following error message: *non-static variable y cannot be referenced from a static context.* Explain why it generates this message.

```
class Test{
  private static int x = 0;
  private int y;
  Test(int kk){y = kk;}
  public static void modX(){x = y;}
}
```

(3 marks)

Total (10 marks)

QUESTION 6

(a) Instances of both the class String and the class Integer are said to be immutable. Explain what this means.

(3 marks)

(b) Given below is the class Circle. Explain why this class is not immutable. What changes must be made to the class to make it immutable? Re-write it by making the necessary changes.

```
class Circle{

    private int x,y;

    private int radius;

    public Circle(int a, int b, int r){

            x = a; y = b; radius = r;

    }

    public int radius(){return radius;}
```

```
    public void move(int a, int b){
        x = x + a; y = y + b;
    }
}
```

(7 marks)

Total (10 marks)

QUESTION 7

(a) The Object class is said to be the base class of all classes. Explain what this means? Name any two methods that are inherited from the Object class.

(3 marks)

(b) Given is the abstract class Employee defined below. It has two abstract methods monthlySalary and monthlyTax.

```
abstract class Employee{
    private String name;
    Employee(String n){name = n;}
    public String name(){return name;}
    public abstract double monthlySalary();
    public abstract double monthlyTax(double rate);
    public String toString(){return name;}
}
```

A manager is an employee that has a yearly salary. Using the abstract class Employee write a class to encapsulate a manager. The monthly salary is calculated by dividing the yearly salary by 12 and tax is calculated by multiplying the monthly salary by the given rate. You must also override the toString method in the Employee class.

(7 marks)

Total (10 marks)

QUESTION 8

(a) Given below is a class Player that has attributes name and team. This class must implement the Comparable interface. Your task is to implement the method for this interface. The ordering is based on player name only.

```
final class Player implements Comparable<Player>{
    private String name;
    private String team;
    Player(String n, String t){
        name = n; team = t;
    }
    Player(String n, String t, int sc){
        name = n; team = t; scored = sc;
    }
    public String name(){return name;}
```

```
    public String team(){return team;}
}
```

(5 marks)

(b) The word final is used before class Player above. What does this word mean?

(2 marks)

(c) Tickets consist of four colours: green, blue, red and white. Each ticket has an associated price: green costs €5, blue costs €10, red costs €20 and white costs €30. The enumerated class Ticket is given. Your task is to write a value method that returns its cost.

```
enum Ticket{
    GREEN,BLUE,RED,WHITE;
    public int value(){ … }
}
```

(3 marks)

Total (10 marks)

QUESTION 9

Given below is a class that represents an account with three attributes: account number *accNum*, *name* and *balance*.

```
class Account{
        private String accNum;
        private String name;
        private double balance;
        Account(String n, String nm, double b){
                accNum = n; name = nm; balance = b;
        }
        String name(){return name;}
        String accNumber(){return accNum;}
        double balance(){return balance;}
        public String toString(){return name+" "+accNum+" "+balance;}
}
```

Your task is to write, for the class Account, the following methods:

equals, that returns true if two instances of the account class have the same account number; **(6 marks)**

hashCode, that returns a hash code for an instance of the class.**(4 marks)**

Total (10 marks)

QUESTION 10

The class Accounts, outlined below, uses a HashSet to manage a collection of Accounts. The public methods of this class are given and your task is to write the code for each of these. The description of the task for each method is described as a comment in the code below.

(Class Account is described in **Question 9** above and you may assume that class Account has an equals and a hashCode method)

```
class Accounts{
        private HashSet<Account> data = new HashSet<>();
        public void add(Account ac){data.add(ac);}
        public boolean search(Account ac){//seach for ac} (2 marks)
        public Account get(String num){
                //return reference to account that has account number num
        } (2 marks)
        public double totalAmount(){
         //calculate total of all balances on account
        } (3 marks)
        public HashSet<Account> balanceInExcess(double x){
                //return set of accounts that have a balance in excess of x
        } (3 marks)
}
```

(See **Appendix A** for a description of Set methods).

(10 marks)

QUESTION 11

The class ListInteger, outlined below, uses an ArrayList to manage a collection of Integer values. The public methods of this class are given and your task is to write the code for each of these. The description of the task for each method is described as a comment in the code below.

```
class ListInteger{
        private ArrayList<Integer> lst;
        ListInteger(){lst = new ArrayList<>();}
        public void add(List<Integer> dt){//append list dt to lst} (2 marks)
        public void addHead(Integer x){//add x at head of list lst} (2 marks)
        public int freq(Integer x){
          //count the frequency of occurrence of x
        } (3 marks)
        public Integer max(){
          //find largest value in list lst
        } (3 marks)
}
```

(See **Appendix A** for a description of List methods).

(10 marks)

QUESTION 12

(a) All the Collection classes are *generic*. What does the term *generic* mean?

(2 marks)

(b) Explain why you might use an ArrayList in preference to an array when you are choosing a data structure for your data.

(3 marks)

(c) Using methods from the Assert class listed in **Appendix B** complete the methods testInteresection and testUnion. Two sets are provided for use in your test code.

```
class SetTest{
  TreeSet<Integer> s1 = new TreeSet<>(Arrays.asList(2,3,4,5));
  TreeSet<Integer> s2 = new TreeSet<>(Arrays.asList(2,4,7));
  @Test
  public void testIntersection(){  … }
  @Test
  public void testUnion(){ … }
}
```

(5 marks)

Total (10 marks)

Appendix A

Constructor	TreeSet<E>() TreeSet<E>(Collection) HashSet<E>() HashSet<E>(Collection)
Insert item, duplicates ignored	add(E x)
Remove item	remove(E x)
Contains item. Used to check if the set contains an instance of x.	boolean contains(E x)
Number of elements in the set.	int size()
Return a string representation of the elements in the set. Only works correctly if instances of the type have a toString() method defined.	toString()
Returns true if set is empty; false otherwise	boolean isEmpty()
Set union	addAll(Collection)
Set intersection	retainAll(Collection)
Set difference	removeAll(Collection)
Subset	boolean containsAll(Collection)
Remove all elements	clear()
Return an array containing all of the elements in the set	Object[] toArray()
Return an iterator over the elements in the set. Used to traverse elements in the set.	Iterator<E> iterator()
Constructor	ArrayList<E>() ArrayList<E>(Collection) LinkedList<E>() LinkedList<E>(Collection)
Append item	add(E x)
Remove item	remove(Object)
Contains item	Boolean contains(Object)
Number of elements	int size()
Convert to string	toString()
Empty set	Boolean isEmpty()
Remove elements	clear()
Retrieve element given index value	E get(int ind);
Insert element at index	add(int ind, E x);
Change element at index	E set(int ind, E x);
Remove element at index	E remove(int ind)
Get index of object	int indexOf(E x);

Additional Methods for LinkedList class	
Add new element at head of list	addFirst(E x)
Return element at head of list	E getFirst()
Remove element at head of list	E removeFirst()

Appendix B
Methods for Assert Class

Method Name	Semantics
assertArrayEquals(boolean[] expecteds, boolean[] actuals)	Asserts that two boolean arrays are equal.
assertArrayEquals(byte[] expecteds, byte[] actuals)	Asserts that two byte arrays are equal.
assertArrayEquals(char[] expecteds, char[] actuals)	Asserts that two char arrays are equal.
assertArrayEquals(double[] expecteds, double[] actuals, double delta)	Asserts that two double arrays are equal.
assertArrayEquals(float[] expecteds, float[] actuals, float delta)	Asserts that two float arrays are equal.
assertArrayEquals(int[] expecteds, int[] actuals)	Asserts that two int arrays are equal.
assertArrayEquals(long[] expecteds, long[] actuals)	Asserts that two long arrays are equal.
assertArrayEquals(Object[] expecteds, Object[] actuals)	Asserts that two object arrays are equal.
assertEquals(double expected, double actual, double delta)	Asserts that two doubles are equal to within a positive delta.
assertEquals(float expected, float actual, float delta)	Asserts that two floats are equal to within a positive delta.
assertEquals(long expected, long actual)	Asserts that two longs are equal.
assertEquals(Object expected, Object actual)	Asserts that two objects are equal.
assertNotEquals(float unexpected, float actual, float delta)	Asserts that two floats are **not** equal to within a positive delta.
assertNotEquals(long unexpected, long actual)	Asserts that two longs are **not** equals.
assertTrue(Boolean condition)	Asserts that condition holds
assertFalse(Boolean condition)	Asserts that condition does not hold.