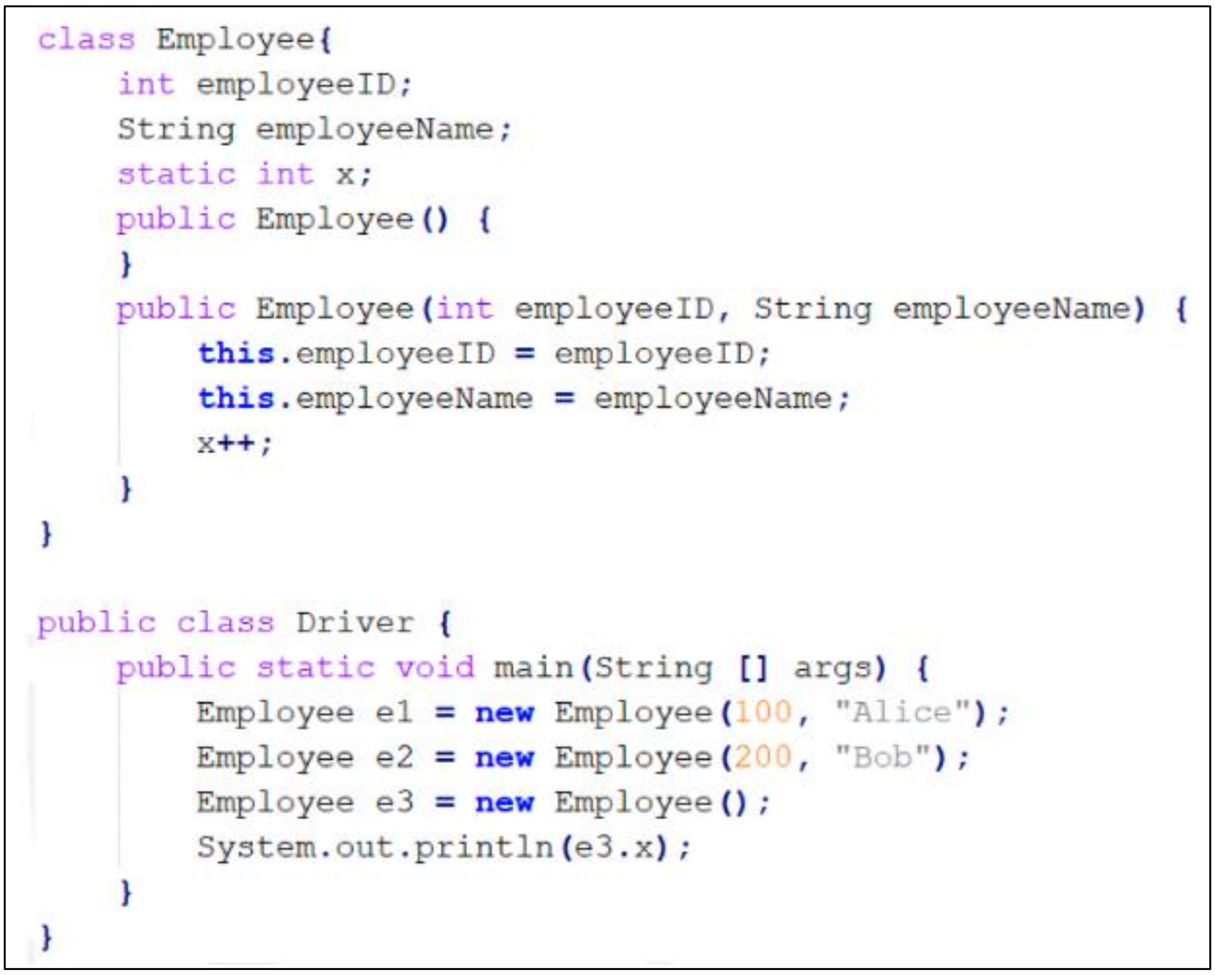
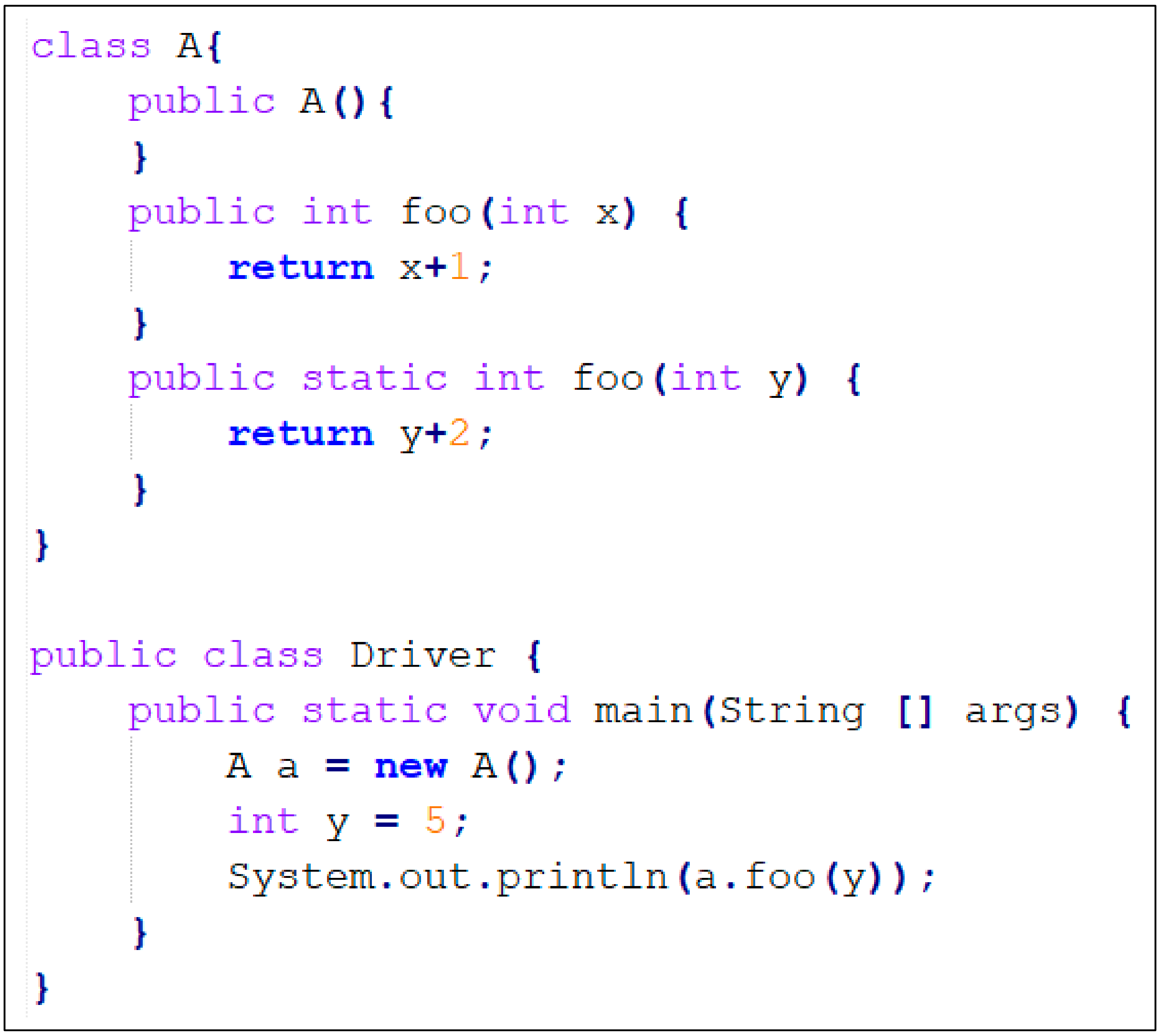
1. What is the output of the following code?
    a. 0
    b. 1
    c. 2
    d. 3
    e. It will not compile



```java
class Employee{
    int employeeID;
    String employeeName;
    static int x;
    public Employee() {
    }
    public Employee(int employeeID, String employeeName) {
        this.employeeID = employeeID;
        this.employeeName = employeeName;
        x++;
    }
}

public class Driver {
    public static void main(String [] args) {
        Employee e1 = new Employee(100, "Alice");
        Employee e2 = new Employee(200, "Bob");
        Employee e3 = new Employee();
        System.out.println(e3.x);
    }
}
```

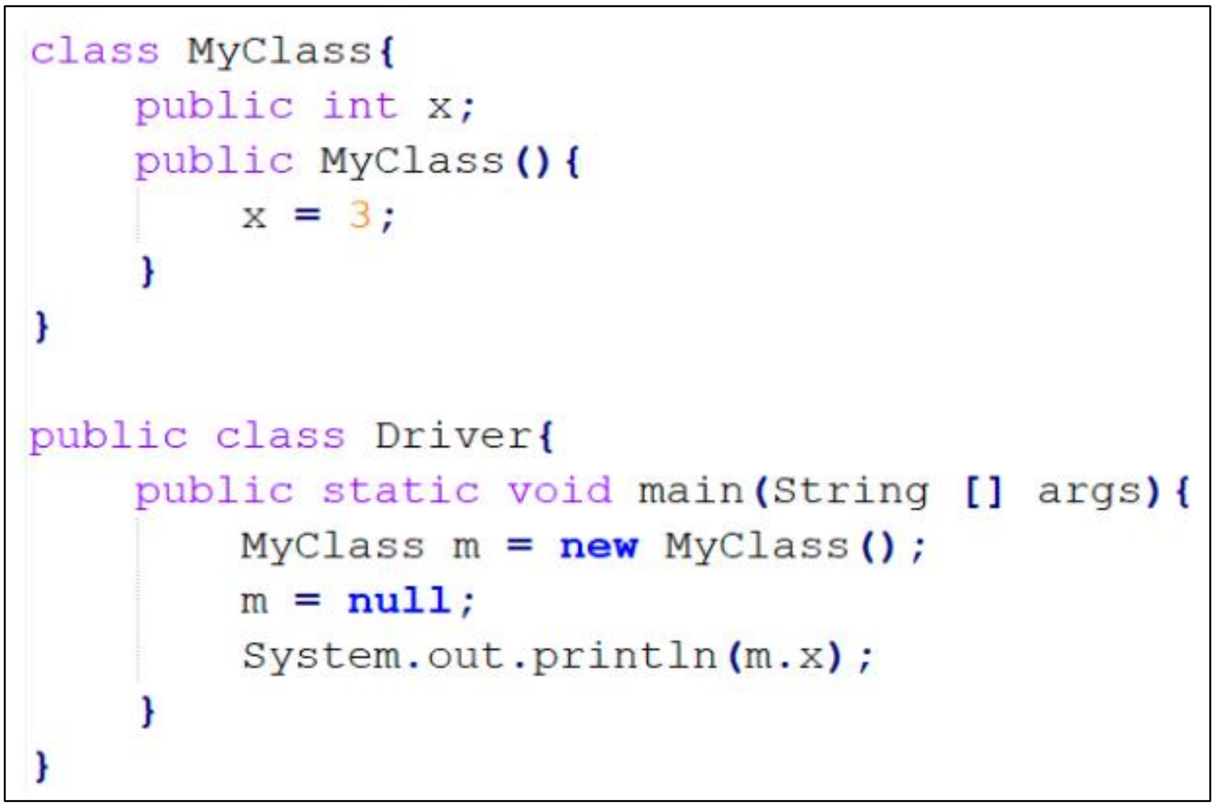
2. What is the output of the following code?
    a. 5
    b. 6
    c. 7
    d. It will not compile



```
class A{
    public A(){
    }
    public int foo(int x) {
        return x+1;
    }
    public static int foo(int y) {
        return y+2;
    }
}

public class Driver {
    public static void main(String [] args) {
        A a = new A();
        int y = 5;
        System.out.println(a.foo(y));
    }
}
```

3. What is the output of the following code?
    a. 0
    b. 3
    c. It will result in a NullPointerException
    d. It will not compile



```
class MyClass{
    public int x;
    public MyClass(){
        x = 3;
    }
}

public class Driver{
    public static void main(String [] args){
        MyClass m = new MyClass();
        m = null;
        System.out.println(m.x);
    }
}
```

4. For each of the following interfaces, indicate whether it is valid or not.

A

```
public interface MyInterface{
    public abstract void foo();
}
```

B

```
public interface MyInterface{
}
```

C

```
public interface MyInterface{
    int x;
    public abstract void foo();
}
```

D

```
public interface MyInterface{
    public void foo(){
    }
}
```

5. For each of the following classes, indicate whether it is valid or not.

A

```
public abstract class MyClass{
    public void foo(){
    }
}
```

B

```
public class MyClass{
    public abstract void foo();
}
```

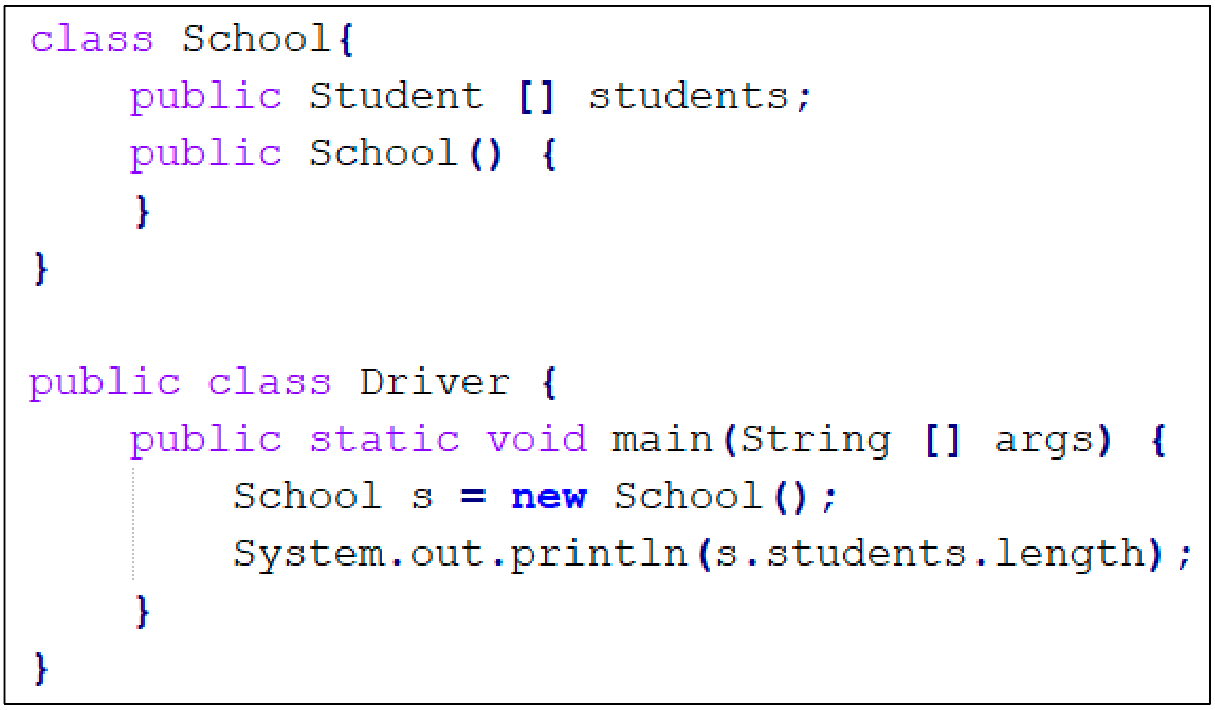
C

```
public abstract class MyClass{
    public abstract void foo();
}
```

D

```
public abstract class MyClass{
    public abstract void foo();
    public void bar(){
    }
}
```

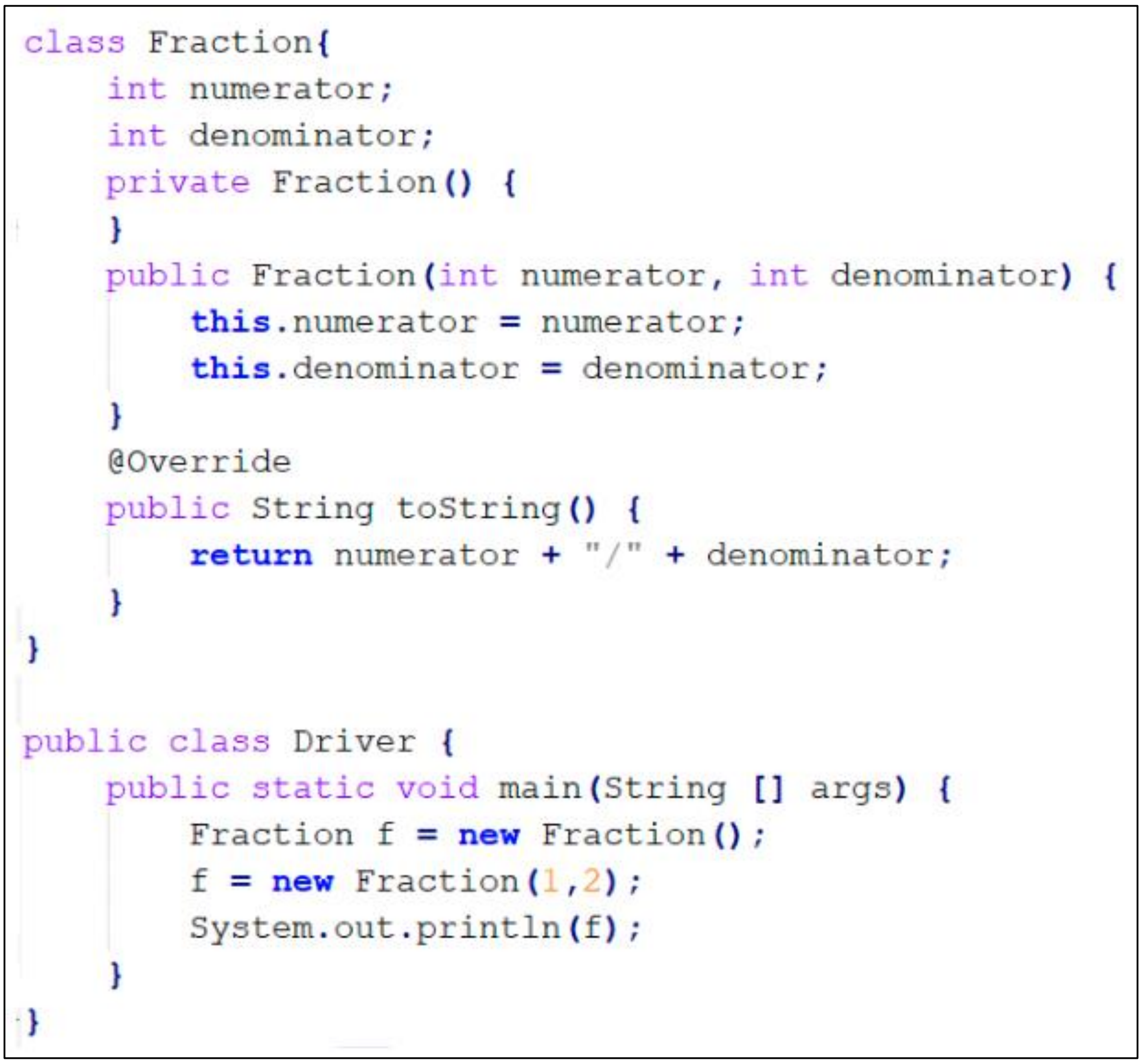
6. What is the output of the following code?
   a. 0
   b. null
   c. It will result in a NullPointerException
   d. It will not compile



```
class School{
    public Student [] students;
    public School() {
    }
}

public class Driver {
    public static void main(String [] args) {
        School s = new School();
        System.out.println(s.students.length);
    }
}
```

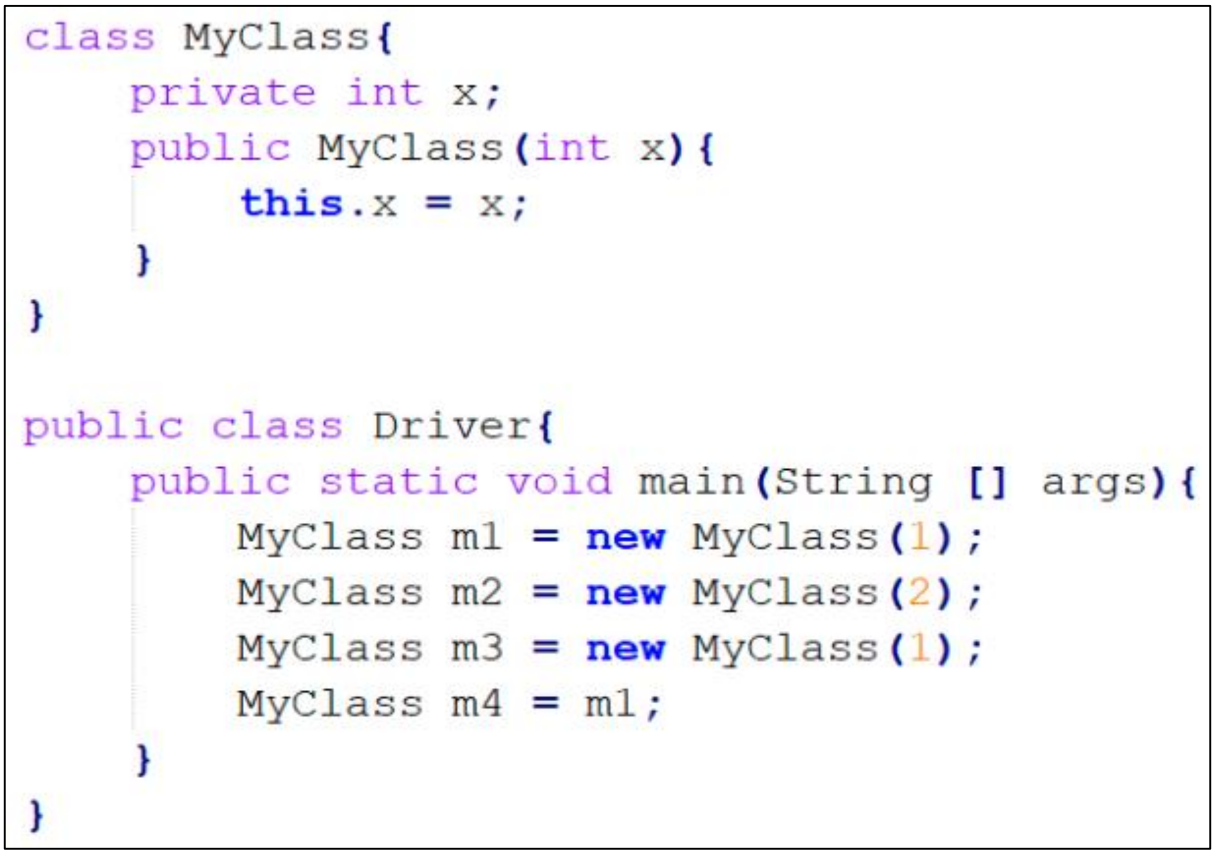
7. What is the output of the following code?
    a. 0.5
    b. 1/2
    c. 0/0
    d. It will result in an ArithmeticException
    e. It will not compile



```java
class Fraction{
    int numerator;
    int denominator;
    private Fraction() {
    }
    public Fraction(int numerator, int denominator) {
        this.numerator = numerator;
        this.denominator = denominator;
    }
    @Override
    public String toString() {
        return numerator + "/" + denominator;
    }
}

public class Driver {
    public static void main(String [] args) {
        Fraction f = new Fraction();
        f = new Fraction(1,2);
        System.out.println(f);
    }
}
```

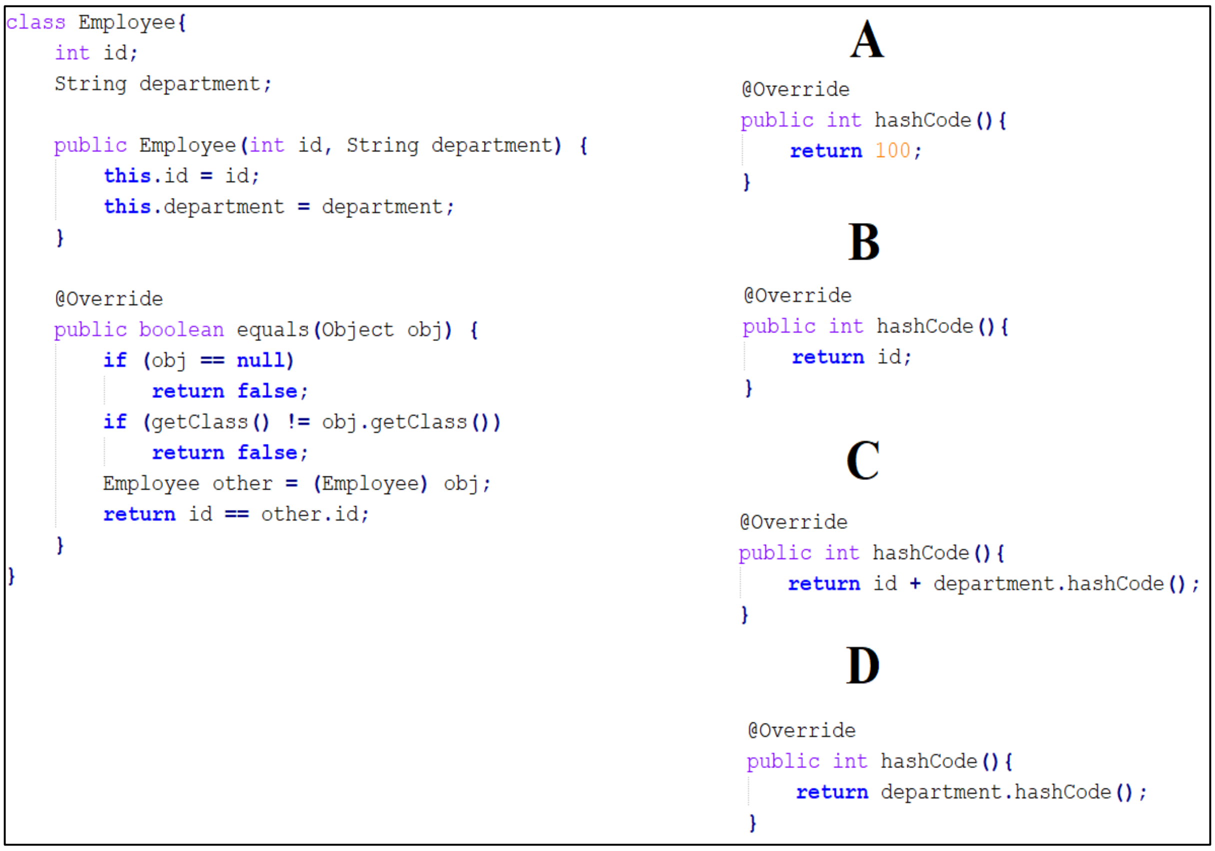
8. What would the result be in each of the following cases?
   a. m1.equals(m2);
   b. m1.equals(m3);
   c. m1.equals(m4);
   d. m1.equals(null);



```
class MyClass{
    private int x;
    public MyClass(int x){
        this.x = x;
    }
}

public class Driver{
    public static void main(String [] args){
        MyClass m1 = new MyClass(1);
        MyClass m2 = new MyClass(2);
        MyClass m3 = new MyClass(1);
        MyClass m4 = m1;
    }
}
```

9. For each of the hashCode implementations, indicate whether it is valid or not. Note that the question is related to the validity of the implementation, not its effectiveness.



```java
class Employee{
    int id;
    String department;

    public Employee(int id, String department) {
        this.id = id;
        this.department = department;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Employee other = (Employee) obj;
        return id == other.id;
    }
}
```

A

```java
@Override
public int hashCode(){
    return 100;
}
```

B

```java
@Override
public int hashCode(){
    return id;
}
```

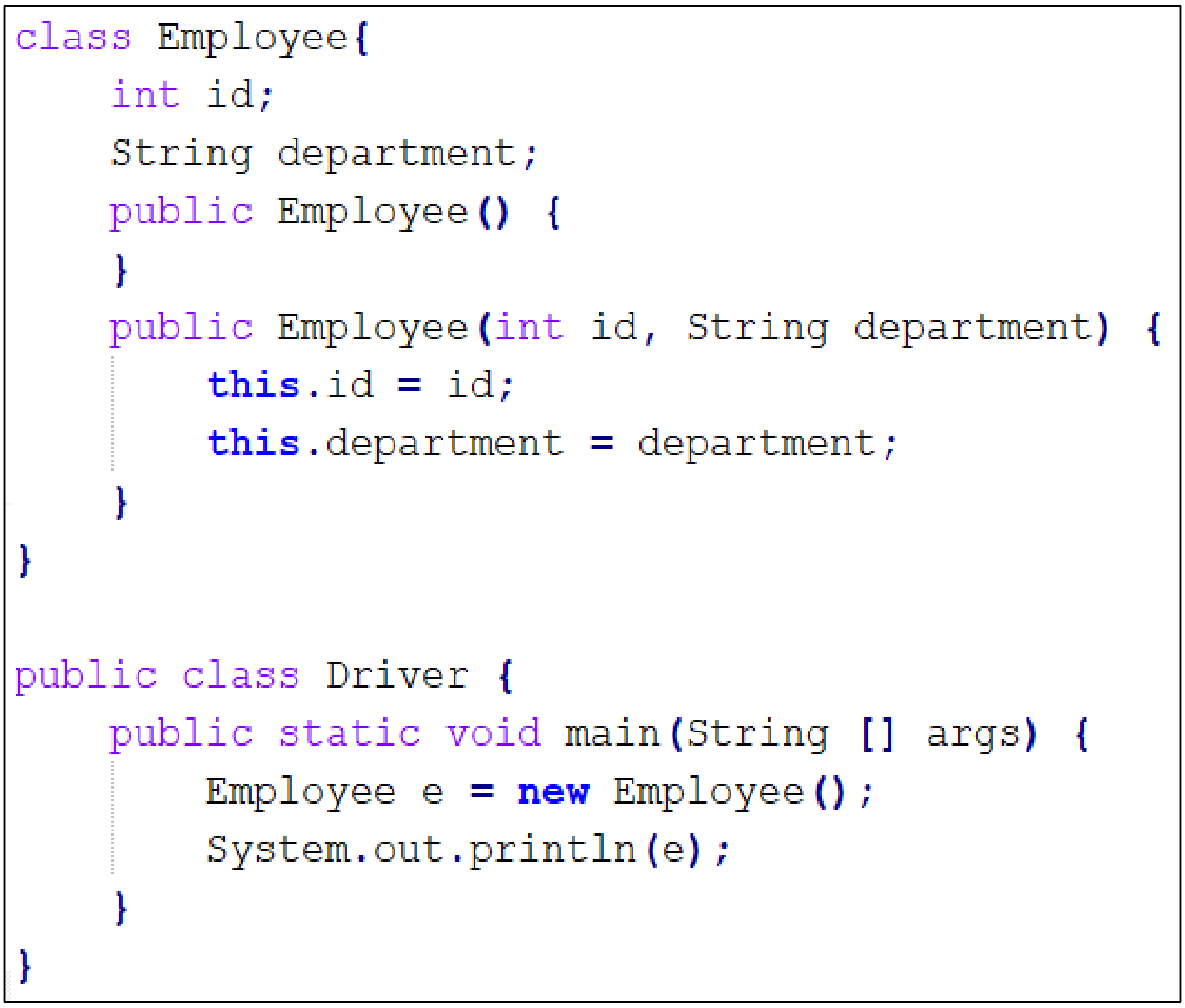
C

```java
@Override
public int hashCode(){
    return id + department.hashCode();
}
```

D

```java
@Override
public int hashCode(){
    return department.hashCode();
}
```

10. What is the output of the following code?
    a. null
    b. 0, null
    c. It will not compile
    d. None of the choices is correct



```
class Employee{
    int id;
    String department;
    public Employee() {
    }
    public Employee(int id, String department) {
        this.id = id;
        this.department = department;
    }
}

public class Driver {
    public static void main(String [] args) {
        Employee e = new Employee();
        System.out.println(e);
    }
}
```

11. What is the output of the following code?

```java
class A{
    public int x;
    public A() {
        x = 1;
    }
    void foo() {
        bar();
    }
    void bar() {
        x++;
    }
}
```

```java
class B extends A{
    public B() {
        x = 5;
    }
    @Override
    void bar() {
        x--;
    }
}
```

```java
public class Driver {
    public static void main(String [] args) {
        A b = new B();
        b.foo();
        System.out.println(b.x);
    }
}
```

12. For each of the following strings, indicate whether it matches the regex **(ab){2,3}[cd]+**
    a. ababccc
    b. ababcdcd
    c. abc
    d. aabbcd

13. For each of the following strings, indicate whether it matches the regex **[^aeiouy]*[aeiouy]+(z|st)***
    a. Quiz
    b. test
    c. east
    d. EAst

14. The regex **[1-9]+[13579]*** matches a set of positive integers. What is this set?
    a. All positive odd numbers
    b. All positive odd numbers greater than 10
    c. All integers greater than zero
    d. None of the choices is correct

15. A Java identifier is a sequence of characters that consists of letters, digits, underscores, and dollar signs. Moreover, an identifier cannot start with a digit. Indicate which of the following regular expressions matches all Java identifiers and only Java identifiers.
    a. [a-zA-Z_$]+[0-9]*
    b. [a-zA-Z_$]?[a-zA-Z0-9_$]*
    c. [a-zA-Z_$][a-zA-Z0-9_$]*
    d. None of the choices is correct.

16. For each of the following strings, indicate whether it matches the regex **\\D+@\\w{2,}.com**
    a. abc@xyz.com
    b. abc@xyzcom
    c. a@@xyz.com
    d. r2d2@naboo.com

17. The following regex matches all strings representing rational numbers whose values are in the range [0,1) and it only matches this set.
    0.[0-9]+
    a. True
    b. False

18. For each of the following regular expressions, indicate whether it matches all strings representing time in 24-hour format and only those strings.
    a. \\d{2}:\\d{2}
    b. [0-2]\\d:[0-6]\\d
    c. ([01][0-9]|2[0-3]):[0-5][0-9]

19. For each of the following strings, indicate whether it matches the regex
    [1-9]\\d{0,2}(,[0-9]\\d{2})*
    a. 1
    b. 100
    c. 1000
    d. 1,000,000

20. Write a regular expression that matches the header of a no-argument java function that returns int or double. For example: int foo(), double bar(), etc.

21. For each of the following aspects of the **equals** method contract, indicate whether it is satisfied by the implementation below.
    a. Reflexivity
    b. Symmetry
    c. Transitivity
    d. Consistency

```
class Duration{
    int minutes;
    int seconds;

    public Duration(int minutes, int seconds){
        this.minutes = minutes;
        this.seconds = seconds;
    }

    @Override
    public boolean equals(Object obj){
        if (obj == null)
            return false;
        if (!(obj instanceof Duration))
            return false;
        Duration other = (Duration) obj;
        return minutes==other.minutes && seconds==other.seconds;
    }
}
```

```
class SpecificDuration extends Duration{
    int milliseconds;

    public SpecificDuration(int minutes, int seconds, int milliseconds){
        super(minutes, seconds);
        this.milliseconds = milliseconds;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;
        if (!(obj instanceof Duration))
            return false;
        if (!(obj instanceof SpecificDuration))
            return super.equals(obj);
        SpecificDuration other = (SpecificDuration) obj;
        return super.equals(other) && milliseconds==other.milliseconds;
    }
}
```

22. Will the following code compile?

```
class Fraction implements Comparable<Fraction>{
    int numerator;
    int denominator;

    public Fraction(int numerator, int denominator){
        this.numerator = numerator;
        this.denominator = denominator;
    }

    @Override
    public int compareTo(Fraction other) {
        double d1 = (double)numerator/(double)denominator;
        double d2 = (double)other.numerator/(double)other.denominator;
        if(d1 < d2)
            return -1;
        else if(d1 == d2)
            return 0;
        return 1;
    }
}
```

```
class UnitFraction extends Fraction{
    public UnitFraction(int denominator){
        super(1,denominator);
    }

    @Override
    public int compareTo(UnitFraction other) {
        double d1 = 1/(double)denominator;
        double d2 = 1/(double)other.denominator;
        if(d1 < d2)
            return -1;
        else if(d1 == d2)
            return 0;
        return 1;
    }
}
```

23. What is the output of the following code?

```
class Fraction{
    int numerator;
    int denominator;

    public Fraction(int numerator, int denominator) throws Exception{
        if(denominator==0)
            throw new Exception("Denominator cannot be zero");
        this.numerator = numerator;
        this.denominator = denominator;
    }
}
public class Driver {
    public static void main(String [] args){
        try{
            Fraction f = new Fraction(1,0);
            System.out.println("try block");
        }
        catch(RuntimeException ex) {
            System.out.println("first catch block");
        }
        catch(Exception ex) {
            System.out.println("second catch block");
        }
        finally{
            System.out.println("finally block");
        }
    }
}
```

24. Does the following code verify the correctness of method **sort**? Explain.

```
@Test
void testSort(){
    int [] A = readArray();
    sort(A);
    boolean isSorted = true;
    for(int i=0;i<A.length-1;i++)
        if(A[i] > A[i+1])
            isSorted = false;
    assertTrue(isSorted);
}
```

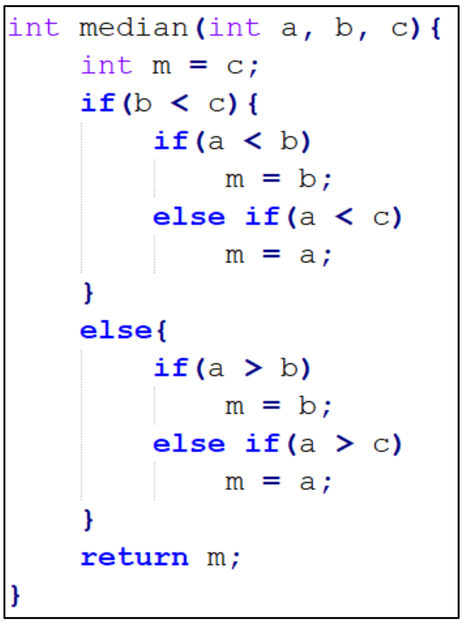
25. The following function is supposed to return the median of three integer values provided as input. However, there is a fault in the code as shown in the highlighted line. For each of the RIPR conditions, indicate whether it is satisfied or not. Assume that the test case is: assertTrue(median(2,1,3)==2);

```
int median(int a, b, c){
    int m = c;
    if(b < c){
        if(a < b)
            m = b;
        else if(a < c)
            m = b; //fault, should be: m = a;
    }
    else{
        if(a > b)
            m = b;
        else if(a > c)
            m = a;
    }
    return m;
}
```

26. Method **sort** is tested using the following test suite:
Test 1: A = {4}
Test 2: A = {5, 4}
For each of the coverage criteria below, indicate whether it is satisfied by the given test suite.
- Node coverage
- Edge coverage

```
void sort(int [] A){
    int N = A.length;
    int i = 1;
    while(i < N) {
        int value = A[i];
        int j = i - 1;
        while (j >= 0 && A[j] > value) {
            A[j + 1] = A[j];
            j = j - 1;
        }
        A[j + 1] = value;
        i++;
    }
    System.out.println("Done");
}
```

27. Given the code below, indicate whether edge coverage is achieved by the following test suite: (1,1,2), (3,2,1), (2,3,1)



```
int median(int a, b, c){
    int m = c;
    if(b < c){
        if(a < b)
            m = b;
        else if(a < c)
            m = a;
    }
    else{
        if(a > b)
            m = b;
        else if(a > c)
            m = a;
    }
    return m;
}
```

28. Given the following code:

```
public int indexOf(int x, int [] A){
        int i=0;
        while(i<A.length){
                if(A[i] == x)
                        return i;
                else
                        i++;
        }
        return -1;
}
```

a. Draw the control flow graph.
b. What is the minimum number of test cases required to satisfy node coverage? Provide an example of such tests.
c. What is the minimum number of test cases required to satisfy edge coverage? Provide an example of such tests.
d. Regarding the control flow graph of part a, is there a test suite that satisfies node coverage but not edge coverage? If yes, provide an example. Otherwise, explain why there is no such test suite.

29. Given a boolean expression and a test suite consisting of three test cases as follows:

Expression: ((a>b) || c) && (x<y)
Test cases: (a=2, b=1, c=false, x=4, y=5), (a=1, b=2, c=false, x=4, y=5), (a=1, b=2, c=true, x=5, y=4)

Indicate whether active clause coverage is satisfied by the test suite. If not, add as many tests as needed to achieve that.

30. Given the same boolean expression as that of question 29, provide a test suite that satisfies clause coverage but not predicate coverage.

31. The following expression checks whether a student is eligible to register in CSCB07:
passedCSCA48 && (cgpa>=3.5 || inCSCPOSt) && !passedCSC207

    Assuming the code is tested using the following test suite, is predicate coverage satisfied? What about clause coverage and active clause coverage?

    Test 1: (passedCSCA48=true, cgpa=3.2, inCSCPOSt=true, passedCSC207=false)
    Test 2: (passedCSCA48=false, cgpa=3.2, inCSCPOSt=true, passedCSC207=false)
    Test 3: (passedCSCA48=true, cgpa=3.2, inCSCPOSt=false, passedCSC207=false)
    Test 4: (passedCSCA48=true, cgpa=3.7, inCSCPOSt=false, passedCSC207=false)
    Test 5: (passedCSCA48=true, cgpa=3.7, inCSCPOSt=false, passedCSC207=true)

32. The code below searches for an integer x in a sorted array A. If x is found, it returns the index of the corresponding location. Otherwise, it returns -1. Provide a test case (i.e. a sorted array of integer and a value to search for) that satisfies Reachability, Infection, and Propagation.

```
int binarySearch(int [] A, int x){
    int left = 0, right = A.length - 1;
    while (left < right) //fault -- should be: while (left <= right)
    {
        int mid = left + (right - 1) / 2;
        if (A[mid] == x)
            return mid;

        if (A[mid] < x)
            left = mid + 1;

        else
            right = mid - 1;
    }
    return -1;
}
```

33. The code below searches for an integer x in a sorted array A. If x is found, it returns the index of the corresponding location. Otherwise, it returns -1. Provide a test case (i.e. a sorted array of integer and a value to search for) that satisfies Reachability and Infection without satisfying Propagation.

```java
int binarySearch(int [] A, int x){
    int left = 0, right = A.length - 1;
    while (left <= right){
        int mid = left + (right - l) / 2;
        if (A[mid] == x)
            return mid;

        if (A[mid] > x) //fault -- should be: if (A[mid] < x)
            left = mid + 1;

        else
            right = mid - 1;
    }
    return -1;
}
```

<u>Programming Exercise 1</u>

Write a Java program that does the following:

1) Reads a folder path provided by the user at runtime using the keyboard

2) Searches all the files in that folder and displays that names of those that contain postal codes. We will assume that postal codes have the following format:

LDL DLD

where L represents an uppercase letter other than 'D', 'F', 'I', 'O', 'Q', and 'U' and D represents a digit between 0 and 9. Note that there is a space between **LDL** and **DLD**.

<u>Programming Exercise 2</u>

You are required to develop and test code that handles academic information related to professors, courses, and students. The steps to be done are as follows:

1. Define a class **Person** as follows:

    a. It has a field of type **int** named **sin** (social insurance number) and another field of type **String** named **name**. Both fields are private.

    b. It has a public constructor that takes one parameter of type **int** and another one of type **String**, and initializes **sin** and **name** accordingly.

    c. It overrides **equals**. Two objects of type **Person** are equal if and only if their **sin** values are equal.

    d. It overrides **hashCode**.

    e. It overrides **toString** by returning **name**

   f. It has a public method **compareName** that takes an object of type **Person** as an argument and returns the result of comparing the name of the calling object with that of the argument. This method should throw an **IllegalArgumentException** if the argument is **null**. Note that you can compare the names using the **compareTo** method of class **String**.

   g. It should be defined such that it would not be possible to instantiate it using the **new** operator.

2. Define class **Professor** as follows:

   a. It inherits from **Person.**

   b. It doesn't have any fields.

   c. It has a public constructor that takes one parameter of type **int** and another one of type **String**, and initializes **sin** and **name** accordingly.

   d. It overrides **toString** by returning the name of the professor as **"Prof. [name]"**. For example, if the name is "X", the returned value would be "Prof. X"

3. Define class **Student** as follows:

   a. It inherits from **Person**.

   b. It has four package-private fields

      i. **cgpa** of type **double** (representing the cumulative grade points average)

      ii. **inCSCPOSt** of type **boolean** (indicating whether the student in enrolled in a CSC subject POSt)

      iii. **passedCSCA48** of type **boolean** (indicating whether the student has passed CSCA48)

      iv. **passedCSC207** of type **boolean** (indicating whether the student has passed CSC207)

   c. It has a public constructor that takes the following arguments and initializes the fields accordingly: **int** (to initialize **sin**), **String** (to initialize **name**), **double** (to initialize **cgpa**), **boolean** (to initialize **inCSCPOSt**), **boolean** (to initialize **passedCSCA48**), **boolean** (to initialize **passedCSC207**)

   d. It overrides **toString** by returning the name and cgpa information as **"[name], cgpa: [cgpa]"**. For example, if the name is "Sam" and the cgpa is 3.6, the returned value would be "Sam, cgpa: 3.6"

   e. It implements **Comparable**. Students should be ordered by **name** and if the names are equal, they should be ordered based on the **cgpa** values (lowest first).

4. Define class **Course** as follows:

   a. It has three package-private fields

      i. **code** of type **String** (representing the code of the course, e.g. "CSCB07")

        ii. **professor** of type **Professor** (representing the instructor of the course)
        iii. **students** of type **ArrayList<Student>** (representing the students registered in the course)

    b. It has a public constructor that takes one parameter of type **String** and another one of type **Professor**, and initializes **code** and **professor** accordingly. It also initializes **students** to an empty **ArrayList<Student>**

    c. It overrides **equals**. Two objects of type **Course** are equal if and only if their **code** values are equal.

    d. It overrides **hashCode**

    e. It has a public method named **isEligibile** that takes an object of type **Student** as an argument and returns a **boolean** value indicating whether the student is eligible to be registered in the course or not. To implement this method, you need to use the CSCB07 eligibility criteria. The student must satisfy the following three conditions:
        i. Passed CSCA48
        ii. cgpa>=3.5 or enrolled in a CSC subject POSt
        iii. Did not pass CSC207

    f. It has a public void method named **addStudent** that takes an object of type **Student** as an argument and adds it to **students** if it satisfies the following conditions:
        i. The student is eligible to be registered in the course.
        ii. The student is not already added to the list of students. You can use method **contains** of class **ArrayList** to check if that is the case.

    g. It has a public void method named **displayInfo** that takes no arguments and displays the course information in the following order: code, professor, students. Note that the list of students should be sorted before being displayed (you can use **students.sort(null);** to achieve that).

5. Write JUnit tests to validate your code according to the following guidelines:
    a. Each test method should validate a specific behavior and should not contain code that is not relevant to the assertion.
    b. One of the test methods should validate the behavior of **compareName** of class **Person** when a **null** value is passed as an argument. That is, the test method should validate that an **IllegalArgumentException** is thrown.