Homework 1

- ECE558
- Nikolay Nikolov
- Winter 2021
- Prof. Roy Kravitz

Question 1:

- a

1. When using the keyword extends, the subclass inherits the private members of its superclass. [F]
2. In Java a static method can access static class variables but a non-static method cannot [F]
3. An interface can contain one or more abstract methods [T]
4. We can instantiate an array by assigning values when the array is declared. [F]
5. The automatic conversion of a Java primitive numeric type to its wrapper class is called autoboxing [T]

- b

[15] Three key tenets of OO programming are encapsulation, inheritance, and overloading. Provide a short definition and describe how they are realized in the Java programming language.

- Encapsulation
    - Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.
    - Declare the variables of a class as private.
    - Provide public setter and getter methods to modify and view the variables values.

  Source: https://www.tutorialspoint.com/java/java_encapsulation.htm
- Inheritance
    - Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
    - extends is the keyword used to inherit the properties of a class

  Source: https://www.tutorialspoint.com/java/java_inheritance.htm
- Overloading
    - If a class has multiple methods having same name but different in parameters, it is known as Method Overloading

  Source: https://www.javatpoint.com/method-overloading-in-java
- c. (Select the best answer)

Interfaces are a special Java concept. Which of the following statements is true about interfaces?

[x] In a class that implements two interfaces, the methods of both interfaces must be implemented.

[] If one of the interface methods in an interface class (defined by ‘interface CLASSNAME’) has a body, all of them must have one.

[] The constructor of an interface allocates the memory for its attributes.

[] In class that implements two interfaces, only the methods of one interface must be implemented.

[] Interfaces can only extend one superinterface because Java does not support multiple inheritance.

Question 2: Java Basics

a)

Source code:

```
/**
 * Nikolay Nikolov ECE558 Winter 2021
 * <h2>Question 2 a)
 * <h2>Write a method that takes an integer input from the user, then prompts
 * for additional integers and prints all of the integers that are greater than
 * or equal to the original input until the user enters a negative number, which
 * is not printed
 */

import java.io.*;
import java.util.*;

public class Homework1 {
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        int[] previousNumbers = new int[100]; // store the user's previous numbers
        int index = 0;
        int originalInput = -1;
        // ---------------------
        while (true){
            System.out.println("Type a number.Negative number will make me exit");
            // Get user input
            int userNumber = Integer.parseInt(scanner.nextLine());
            // if the number is negative exit
            if(userNumber < 0 ){
                System.out.println("Goodbye!");
                System.exit(-1);
            }
            else {
                // The very first time the App is running
                if (originalInput < 0){
                    originalInput = userNumber;
                }
                // Store numbers in array
                previousNumbers[index] = userNumber;
                // Print numbers >= originalInput
                for (int i = 0; i < previousNumbers.length; i++)
                {
                    if (previousNumbers[i] >= originalInput )
                    {
                        System.out.println("[ " + previousNumbers[i] + ">=" + originalInput + " ]");
                    }
                }
                // increment index for the array of previous numbers
                index = index + 1;
            }
        }
        // ------------------------
        // While loop ends here
    }

}
```

Traces from testing:

```
 [niko@toolbox homework1]$ java Homework1
Type a number.Negative number will make me exit
9
[ 9>=9 ]
Type a number.Negative number will make me exit
1
[ 9>=9 ]
Type a number.Negative number will make me exit
0
[ 9>=9 ]
Type a number.Negative number will make me exit
10
[ 9>=9 ]
[ 10>=9 ]
Type a number.Negative number will make me exit
3
[ 9>=9 ]
[ 10>=9 ]
Type a number.Negative number will make me exit
11
[ 9>=9 ]
[ 10>=9 ]
[ 11>=9 ]
Type a number.Negative number will make me exit
12
[ 9>=9 ]
[ 10>=9 ]
[ 11>=9 ]
[ 12>=9 ]
Type a number.Negative number will make me exit
13
[ 9>=9 ]
[ 10>=9 ]
[ 11>=9 ]
[ 12>=9 ]
[ 13>=9 ]
Type a number.Negative number will make me exit
-1
Goodbye!
```

b)

1) The following code sequence is intended to print Hello three times; however, it

only prints Hello once. Where is the problem in this code sequence?

Original Code

```

public static void main(String[] args);
for (int i = 0; i < 3; i++){
    System.out.println("Hello");

}
```

- Removing the semicolon. With the simicolon, it runs 3 loops and then prints

Modified Code

```
public class Homework1_2b_1 {

    public static void main(String[] args) {
        for (int i = 0; i < 3; i++){
            System.out.println("Hello");
        }

    }

}
```

Traces:

```
[niko@toolbox homework1]$ java Homework1_2b_1
Hello
Hello
Hello
[niko@toolbox homework1]$
```

2) You coded the following in class Hw1.java:

int a = 32, b = 10; double c = a / b; System.out.println(“The value of c is “ + c); You expected the value of c to be 3.2, but instead c was displayed as 3. Explain what the problem is and write the code to fix it.

The problem is that a and b are integers

Correct code:

```
 public static void main(String[] args) {
    double a = 32, b = 10;
    double c = a / b;
    System.out.println("The value of c is " + c);

}
```

Traces:

```
 [niko@toolbox homework1]$ java Homework1_2b_2
The value of c is 3.20
```

3)

You coded the following: int[][] a = {{9,8,7,6},{10,20,30,40}}; for (int j = 0, j <= a[1].length; j++) { if (a[1][j] == 20) { System.out.println(“Found 20 at column index “ + j + + “ of second row”); } } The code compiles properly, but when you run the program you get the following output: Found 20 at column index 1 of second row Exception in thread “main” java.lang.ArrayIndexOutOfBoundsException: 4 at Test.main(Test.java:14)

- In order the code to work we need to remove the '[1]'. Since the length of a is 2 not 4.

```
 Correct Code:

public static void main(String[] args) {
    int[][] a = {{9,8,7,6},{10,20,30,40}};
    for (int j = 0; j <= a.length; j++) {
        System.out.println("a[1].length is" + a[1].length);
        System.out.println("a.length is" + a.length);

        if (a[ 1 ][j] == 20) {
            System.out.println("Found 20 at column index " + j + " of second row");
        }
    }

}
```

Traces:

```
[niko@toolbox homework1]$ java Homework1_2b_3
a[1].length is4
a.length is2
a[1].length is4
a.length is2
Found 20 at column index 1 of second row
a[1].length is4
a.length is2
```

Question 3:

Source Code

```
 Rational.java

[niko@toolbox homework1]$ cat Rational.java
/**
 * Class Rational.
 * Nikolay Nikolov
 * ECE558 Winter 2021
 *
 */

class Rational {

  /*
   * This is the object
   * for the rational number
   */
  // ---------------------------------
  /**
   * Private attribute numerator for class Rational.
   * this is the numerator
   */
  private double num;

  /**
   * Private attribute denominator for class Rationa.
   */
  private double den = 1;

  // constructor
  Rational() {
    // First constructor
  }

  // constructor
  Rational(final double numerator, final double denominator) {
    if (denominator == 0) {
      System.err.println("Error.Denominator cannot be zero");
    }
    this.num = numerator;
    this.den = denominator;
  }

  // Getter Numerator
  public double getNumerator() {
    return num;
  }

  // Setter Numerator
  public double setNumerator(final double newNumerator) {
    this.num = newNumerator;
    return this.num;
  }

  // Getter Denominator
  public double getDenominator() {
    return den;
  }

  // Setter Denominator
  public double setDenominator(final double newDenominator) {
    this.den = newDenominator;
    return this.den;
  }

  // Equals()
  public boolean equals() {
    final double frac = 0.001;
    if (frac == (Math.abs(num / den))) {
      return true;
    }
    return false;
  }

  // toString()
  public String toString() {
    StringBuffer buf = new StringBuffer();
    buf.append("Numerator   = ");
    buf.append(num);
    buf.append("\n");
    buf.append("Denominator = ");
    buf.append(den);
    buf.append("\n");
    return buf.toString();
  }

  // multiplication of two rational numbers
  public double multiply() {
    return (num * den);
  }

  // addition of two rational numbers
  public double sum() {
    return (num + den);
  }
}
```

```
}
// End
```

Testing the Rational.java

```
[niko@toolbox homework1]$ cat ClientRational.java
/**
 * Client for Rational class.
 * Nikolay Nikolov
 * ECE558 Winter 2021
 * It has a test runner
 * that make simple comparison
 * assertion like, but assertion is normally
 * locked in Java
 */

class TestRunner {

  /*
   * Simple TestRunner.
   */

  /**
   * Returned value from a function call.
   */
  private double returnedValue;

  /**
   * Expected value if the function returns correctly.
   */

  private double expectedValue;
  /**
   * Test index is the test count.
   */

  private int testIndex;
  /**
   * Test name is the description of the test.
   * Example. function A should return B when input is C
   */

  private String testName;

  /**
   * Constructor for the TestRunner.
   *
   * @param name - test name
   * @param ret - returned value from function tested
   * @param exp - expected value from function tested
   * @param index - test index
   */
  TestRunner(
    final String name,
    final double ret,
    final double exp,
    final int index
  ) {
    this.testName = name;
    this.returnedValue = ret;
    this.expectedValue = exp;
    this.testIndex = index;
  }
  // function to run the test
  public void runner() {
    if (returnedValue == expectedValue) {
      System.out.println("Test: " + testIndex);
      System.out.println(testName);
      System.out.println("Success");
      System.out.println("---------------");
    } else {
      System.out.println("Test: " + testIndex);
      System.out.println(testName);
      System.out.println("Fail");
      System.out.println("---------------");
    }
  }
}

public class ClientRational extends Rational {

  /**
   * Main.
   * Bellow are the tests
   * @param args - command line input
   */
  public static void main(final String[] args) {
    // Test#1
    final int numer = 1;
    final int denom = 5;
    Rational ratObj = new Rational(numer, denom);
    double sum = ratObj.sum();
    final int expectedSum = 6;
    int testIndex = 1;
    TestRunner runTest1 = new TestRunner(
      "sum should return 6 when num = 1 and den = 5",
      sum,
      expectedSum,
      testIndex
    );
    runTest1.runner();
    // Test#2
    double multiply = ratObj.multiply();
    final int expectedMultiply = 5;
    testIndex = 2;
    TestRunner runTest2 = new TestRunner(
      "multiply should return 5 when num = 1 and den = 5",
      multiply,
      expectedMultiply,
      testIndex
    );
    runTest2.runner();
    // Test#3
    String returnedString = ratObj.toString();
    String expectedString = "Numerator   = 1.0\n" + "Denominator = 5.0\n";
    if (returnedString.equals(expectedString)) {
      System.out.println("Test: 3");
      System.out.println("Testing toString()");
      System.out.println(returnedString);
      System.out.println("Success");
      System.out.println("---------------");
    }
    // Test#4
    // Test Setter and Getter for Numerator
    final double newNumerator = 15;
    ratObj.setNumerator(newNumerator);
    final int expectedNumerator = 15;
    String expectedNewString = "Numerator   = 15.0\n" + "Denominator = 5.0\n";
    String expectedNewString = "Numerator   = 15.0\n" + "Denominator = 5.0\n";
    String returnedNewString = ratObj.toString();
    if (returnedNewString.equals(expectedNewString)) {
      System.out.println("Test: 4");
      System.out.println("Testing setting the Numerator to 15");
      System.out.println(returnedNewString);
      System.out.println("Success");
      System.out.println("---------------");
    }
    // Test#5
    // Test Setter and Getter for Denominator
    final double newDenominator = 20;
    ratObj.setDenominator(newDenominator);
    final int expectedDenominator = 20;
    final String expectedNewDenominator =
      "Numerator   = 15.0\n" + "Denominator = 20.0\n";
    final String returnedNewDenominator = ratObj.toString();
    if (returnedNewDenominator.equals(expectedNewDenominator)) {
      System.out.println("Test: 5");
      System.out.println("Testing setting the Denominator to 20");
      System.out.println(returnedNewDenominator);
      System.out.println("Success");
      System.out.println("---------------");
    }
  }
}
// End
```

Traces from testing

```
[niko@toolbox homework1]$ java ClientRational
Test: 1
sum should return 6 when num = 1 and den = 5
Success
---------------
Test: 2
multiply should return 5 when num = 1 and den = 5
Success
---------------
Test: 3
Testing toString()
Numerator   = 1.0
Denominator = 5.0

Success
---------------
Test: 4
Testing setting the Numerator to 15
Numerator   = 15.0
Denominator = 5.0

Success
---------------
Test: 5
Testing setting the Denominator to 20
Numerator   = 15.0
Denominator = 20.0

Success
---------------
[niko@toolbox homework1]$
```

Question 4

Source code

Game.java

```
/**
 * Provided for HW1.
 */

public class Game {

  /**
   * Attribute.
   */
  private String mDescription;

  /**
   * Constructor.
   * @param description - String description of the game
   */
  public Game(final String description) {
    setDescription(description);
  }

  Game() {
    //
  }

  /**
   * Getter for Description.
   */
  public String getDescription() {
    return mDescription;
  }

  /**
   * Setter for Description.
   */
  public void setDescription(String description) {
    mDescription = description;
  }

  public String toString() {
    return ("description: " + mDescription);
  }
}
[niko@toolbox homework1]$
```

BoardGame.java

```
/**
 * Board Game class that inherits from Game.
 * HW 1 Q4
 */

class BoardGame extends Game {

  /**
   * Constructor.
   * @param description - String for the super class
   * @param number - number of players
   * @param tie - String [yes/no] to allow tie
   * @param min - int min number of users
   * @param max - int max number of users
   */
  BoardGame(
    final String description,
    final int number,
    final String tie,
    final int min,
    final int max
  ) {
    super(description);
    if (number == 0 || number == 1) {
      System.err.println("Error.Cannot play with 0 or 1 players");
    }
    this.numberPlayers = number;
    this.allowTie = tie;
    this.minNum = min;
    this.maxNum = max;
  }

  BoardGame() {
    //
  }

  /**
   * Attribute for the number of players.
   */
  private int numberPlayers;

  /**
   * Attribute for the max number of players.
   */
  private int maxNum;

  /**
   * Attribute for the min number of players.
   */
  private int minNum;

  /**
   * Attribute for whethere the game can end in tie.
   */
  private String allowTie;

  /**
   * Setter for players.
   * @param players - number of players
   */
  public void setPlayers(final int players) {
    this.numberPlayers = players;
  }

  /**
   * Getter for players.
   * @return int
   */
  public int getPlayers() {
    return numberPlayers;
  }

  /**
   * Setter for allowTie.
   * @param tie - string [yes/no] for to allow a tie
   */
  public void setAllowTie(final String tie) {
    this.allowTie = tie;
  }

  /**
   * Getter for allowTie.
   * @return String
   */
  public String getAllowTie() {
    return allowTie;
  }

  /**
   * overwritting toString().
   * @return String
   */
  public String toString() {
    return super.toString();
  }
}
[niko@toolbox homework1]$
```

FunGame.java

```
[niko@toolbox homework1]$ cat FunGame.java
/**
 * Class that uses the BoardGame class.
 * HW 1 Q4
 */

class FunGame extends BoardGame {

  /**
   * Attribute for the min number of players.
   */
  private int min;

  /**
   * Attribute for the max number of players.
   */
  private int max;

  /**
   * Attribut for the time limit.
   */
  private int time;

  /**
   * Attribut for numger of players.
   */
  private int num;

  /**
   * Constructor.
   * @param description - String for the super class
   * @param number - number of players
   * @param tie - String [yes/no] to allow tie
   * @param minNum - int minimum number of players
   * @param maxNum - int maximum number of players
   * @param limitTime - int for time limit to finish the game
   */
```

```
FunGame(
  final String description,
  final int number,
  final String tie,
  final int minNum,
  final int maxNum,
  final int limitTime
) {
  super(description, number, tie, minNum, maxNum);
  if (number > maxNum) {
    System.err.println("Error. Max num exceeded");
  }
  if (number < minNum) {
    System.err.println("Error. Not enough players");
  }

  this.min = minNum;
  this.max = maxNum;
  this.time = limitTime;
  this.num = number;
}

FunGame() {
  //
}

/**
 * Setter for min \# of players.
 * @param minPlayers - number of players
 */
public void setMinPlayers(final int minPlayers) {
  if (minPlayers < num) {
    System.err.println("Error. No enough players");
  } else {
    this.min = minPlayers;
  }
}

/**
 * Getter for min \# of players.
 * @return int
 */
public int getMinPlayers() {
  return min;
}

/**
 * Setter for max \# of players.
 * @param maxPlayers - number of players
 */
public void setMaxPlayers(final int maxPlayers) {
  if (num > maxPlayers) {
    System.err.println("Error. Too many players");
  } else {
    this.max = maxPlayers;
  }
}

/**
 * Getter for max \# of players.
 * @return int
 */
public int getMaxPlayers() {
  return max;
}
```

```
  }

  /**
   * Setter for time limit.
   * @param limitTime - int time in minutes
   */
  public void setTime(final int limitTime) {
    this.time = limitTime;
  }

  /**
   * Getter for time limit.
   * @return int
   */
  public int getTime() {
    return time;
  }

  /**
   * overwritting toString().
   * @return String
   */
  public String toString() {
    return super.toString();
  }
}
[niko@toolbox homework1]$
```

Gamer.java - Client for of the above classes

```
[niko@toolbox homework1]$ cat Gamer.java
/*
 * Client for FunGame,BoarGame,Game.
 * Nikolay Nikolov
 * ECE558 Winter 2021
 * locked in Java
 */

class TestRunnerGame {

  /*
   * Simple TestRunner.
   */

  /**
   * Returned value from a function call.
   */
  private double returnedValue;

  /**
   * Expected value if the function returns correctly.
   */

  private double expectedValue;
  /**
   * Test index is the test count.
   */

  private int testIndex;
  /**
   * Test name is the description of the test.
   * Example. function A should return B when input is C
   */
```

```java
  private String testName;

  /**
   * Constructor for the TestRunner.
   *
   * @param name - test name
   * @param ret - returned value from function tested
   * @param exp - expected value from function tested
   * @param index - test index
   */
  TestRunnerGame(
    final String name,
    final double ret,
    final double exp,
    final int index
  ) {
    this.testName = name;
    this.returnedValue = ret;
    this.expectedValue = exp;
    this.testIndex = index;
  }

  // function to run the test
  public void runner() {
    if (returnedValue == expectedValue) {
      System.out.println("Test: " + testIndex);
      System.out.println(testName);
      System.out.println("Success");
      System.out.println("---------------");
    } else {
      System.out.println("Test: " + testIndex);
      System.out.println(testName);
      System.out.println("Fail");
      System.out.println("---------------");
    }
  }
}

public class Gamer extends FunGame {

  /**
   * Main.
   * Bellow are the tests
   * @param args - command line input
   */
  public static void main(final String[] args) {
    /*
     * Game attributes:
     *  - description
     *  Board Game attributes:
     *  - number of players
     *  - allow tie [yes/no]
     * Fun Game attributes
     *  - min number of players
     *  - max number of players
     *  - time limit for the game
     */

    final String description = "New board game";
    final int players = 4;
    final String allow = "yes";
    final int minNumber = 2;
    final int maxNumber = 4;
    final int time = 1;
    FunGame playGame = new FunGame(
      description,
      players,
      allow,
      minNumber,
      maxNumber,
      time
    );

    // Test#1
    final String returnedDesc = playGame.toString();
    final String expectDesc = "description: New board game";
    if (expectDesc.equals(returnedDesc)) {
      System.out.println("Test: 1");
      System.out.println("Testing setting the description");
      System.out.println(returnedDesc);
      System.out.println("Success");
      System.out.println("---------------");
    }
    // Test#2
    int testIndex = 2;
    final int returnedMin = playGame.getMinPlayers();
    TestRunnerGame runTest1 = new TestRunnerGame(
      "getMinPlayers should return 2",
      returnedMin,
      minNumber,
      testIndex
    );
    runTest1.runner();

    // Test#3
    final int index = 3;
    final int returnedMax = playGame.getMaxPlayers();
    TestRunnerGame runTest2 = new TestRunnerGame(
      "getMaxPlayers should return 4",
      returnedMax,
      maxNumber,
      index
    );
    runTest2.runner();

    // Test#4
    final int indexTest = 4;
    final int returnedTime = playGame.getTime();
    TestRunner runTest3 = new TestRunner(
      "getTime should return 20",
      returnedTime,
      time,
      indexTest
    );
    runTest3.runner();

    // Test#5
    final String returnedAllowTie = playGame.getAllowTie();
    if (returnedAllowTie.equals(allow)) {
      System.out.println("Test: 5");
      System.out.println("getAllowTie should return 'yes'");
      System.out.println(returnedAllowTie);
      System.out.println("Success");
      System.out.println("---------------");
    }

    // Test#6
    final String setToNo = "no";
    playGame.setAllowTie(setToNo); // set to no
    final String returnedAllowTieNo = playGame.getAllowTie();
    if (returnedAllowTieNo.equals(setToNo)) {
      System.out.println("Test: 6");
      System.out.println("setAllowTie should return 'no'");
      System.out.println(returnedAllowTieNo);
      System.out.println("Success");
      System.out.println("---------------");
    }
    // Test#7

    final String secondDescription = "Another board game";
    final int secondPlayers = 10;
    final String allowTie = "no";
    final int secondMinNumber = 2;
    final int secondMaxNumber = 4;
    final int secondTime = 10;

    FunGame newGame = new FunGame(
      secondDescription,
      secondPlayers,
      allowTie,
      secondMinNumber,
      secondMaxNumber,
      secondTime
    );

    final String newString = newGame.toString();
    final int getNewPlayers = newGame.getPlayers();
    System.out.println("Test: 7");
    System.out.println("It should print error for exceeding max num");
    System.out.println(
      "New Players " + getNewPlayers + " > " + secondMaxNumber + " Max Num"
    );

    System.out.println("Success");
    System.out.println("---------------");
  }
}
// End
[niko@toolbox homework1]$
```

Traces :

```
[niko@toolbox homework1]$ javac -d . *.java
[niko@toolbox homework1]$ java Gamer
Test: 1
Testing setting the description
description: New board game
Success
---------------
Test: 2
getMinPlayers should return 2
Success
---------------
Test: 3
getMaxPlayers should return 4
Success
---------------
Test: 4
getTime should return 20
Success
---------------
Test: 5
getAllowTie should return 'yes'
yes
Success
---------------
Test: 6
setAllowTie should return 'no'
no
Success
---------------
Error. Max num exceeded
Test: 7
It should print error for exceeding max num
New Players 10 > 4 Max Num
Success
---------------
[niko@toolbox homework1]$
```