**BBM102 – Introduction to Programming II – Spring 2020**

**Final Exam – The Second Session**

**July 1, 2020, 10:40-12:20**

**First name / Last name:**

**Student ID:**

1. Please answer the following true/false questions. Write a brief explanation (a few sentences) for each question. ***(24 points)***
2. Generic type information is available to the JVM at runtime.

|  |
| --- |
| False. The generics exist only at compile time, and only to enforce things like parametric polymorphism. |

1. If bounded parameters are employed, this means of restricting the types that can be used as generic type arguments.

|  |
| --- |
| True. String cannot be used if parameter is int |

1. An arithmetic exception such as division by zero can be avoided by careful programming while an I/O exception such as file not found may occur regardless of the precautions taken by the programmer.

|  |
| --- |
| True. If the program does not catch I / O exception, it may result in an error |

1. A method that contains a try-catch-finally structure may also have a “throws” declaration.

|  |
| --- |
| True. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. |

1. Lets assume that you create an object x of the Object class. You can assign any object (objects of other classes) to the object x directly (without explicit type casting).

|  |
| --- |
| False. Not directly, need inheritance |

1. We can create an object of a class which is extended from an abstract class without implementing its abstract methods, if we are not going to use these abstract methods.

|  |
| --- |
| False. Abstract methods implementing is mandatory. |

1. We cannot create an object of interfaces in Java. So a variable of an interface type cannot hold a reference to an object of a class that implements this interface.

|  |
| --- |
| True. An interface cannot contain object fields. The only fields that can appear in an interface must be declared both static and final. |

1. A class which is derived from another class cannot be the superclass of any class.

|  |
| --- |
| False. Possible if a class which is derived from another class extends any class. |

1. An interface is able to involve the following type of members: public, static, final fields; static methods along with their bodies.

|  |
| --- |
| True. interface method print; only public, private, abstract, default, static and strictfp are permitted. |

1. Protected members are accessible outside the package if they have been inherited.

|  |
| --- |
| True. It is the inheritance mechanism in java by which one class is allow to inherit the features(fields and methods) of another class. |

1. Finalized methods cannot be overridden.

|  |
| --- |
| False. finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanup. |

1. Many classes are able to implement the same interface.

|  |
| --- |
| False. A Java class can only extend one parent class. Multiple inheritance (extends) is not allowed. Interfaces are not classes, however, and a class can implement more than one interface. |

**3.**  Lets assume that the following Node and List classes are given to you. The list class contains a group of Node objects and holds the beginning of the linked list in order to traverse the list (access each element in the list). Each node has two links apart from storing its data, one link points to the previous node in the list, and the second link points to the next node in the list. According to these definitions, write the following two methods: *insertAtEnd*, *insertBefore*. *insertAtEnd* inserts a new node at the end of the list. *insertBefore* method inserts a new node before the given previous node (preNode in the figure) in the list. An example for *insertBefore* is given below. While implementing the *insertBefore* method, for simplicity, please assume that the list is not empty and preNode refers to an intermediate element in the list. Please complete all blanks (\_\_\_\_\_\_\_\_\_\_) in the code. ***(28 points)***

7

6

3

preNode

9

head

4

4

7

3

9

preNode

6

newNode

head

**public** **class** Node{

**public** **int** data;

**public** Node next;

**public** Node prev;

**public** Node(**int** data){

**this**.data = data;

**this**.next = **this**.prev = **null**;

}

}

**public** **class** List

{

Node head;

**public** List()

{

head = **null**;

}

**public** **void** insertAtEnd( Node newNode){

Node last = head; /\* used in step 5/

/ 3. This new node is going to be the last node, so make next of it as NULL/ newNode.next = null; / 4. If the Linked List is empty, then make the new \* node as head /

if (head == null) {

newNode.prev = null;

head = newNode;

return;

} / 5. Else traverse till the last node /

while (last.next != null)

last = last.next; / 6. Change the next of last node /

last.next = newNode; / 7. Make last node as previous of new node \*/

newNode.prev = last;

}

**public** **void** insertBefore( Node preNode, Node newNode)

{

// if the node is to be inserted before head

if(preNode==head) {

newNode.prev=null;

newNode.next=head;

head.prev=newNode;

head=newNode; return;

}

// insert before

newNode.prev=preNode.prev;

newNode.next=preNode;

preNode.prev.next=newNode;

preNode.prev=newNode;

}

}

1. Please write the Score class in order to produce the following output. Please complete all blanks (\_\_\_\_\_\_\_\_\_\_) in the code. ***(25 points)***

**Bill's test score was less than Sevil's test score.**

**public** **class** Program

{

**public** **static** **void** main(String[] args)

{

**try**{

Score bill = **new** Score("Bill Gates", 84);

Score sevil = **new** Score("Sevil Sen", 91);

//Determine how bob compares to mary.

//It throws exception if different types of objects are to be compared

**int** comparison = bill.compareTo(sevil);

// Print out the results of the comparison.

System.***out***.print("Bill's test score was ");

**if** (comparison == -1)

System.***out***.print("less than ");

**else** **if** (comparison == 0)

System.***out***.print("equal to ");

**else**

System.***out***.print("greater than ");

System.***out***.println("Sevil's test score.");

}

**catch** (Exception ex) {

System.***out***.println(ex.toString());

}

}

}

**public class** Score **implements** Comparable<Skore>{

String string;

int value;

public Score(String string, int value) {

this.string = string;

this.value = value;

}

@Override public int compareTo(Score o) {

If (this.value > o.value) { return 1; }

else if (this.value < o.value){ return -1; }

return 0; }

}