EECS 1022 3.0 Programming for Mobile Computing

Solution of Final Exam

14:00–15:30 on August 4, 2017

1 (2 marks)

(a) **Explain** why the expression `2.0 + 1.0 / 2.0` evaluates to `2.5`.

Answer: Division binds stronger than addition, so the expression is equivalent to `2.0 + (1.0 / 2.0)`. Hence, the expression `2.0 + 1.0 / 2.0` evaluates to 2.5.

Marking scheme: 1 mark for the mention that the division happens first.

(b) **Explain** why the expression `2 + 1 / 2` evaluates to `2`.

Answer: Division binds stronger than addition, so the expression is equivalent to `2 + (1 / 2)`. Since 1 and 2 are integers, the division returns an integer. In Java, integer division rounds down. Hence, `1 / 2` equals 0. Therefore, the expression `2 + 1 / 2` evaluates to 2.

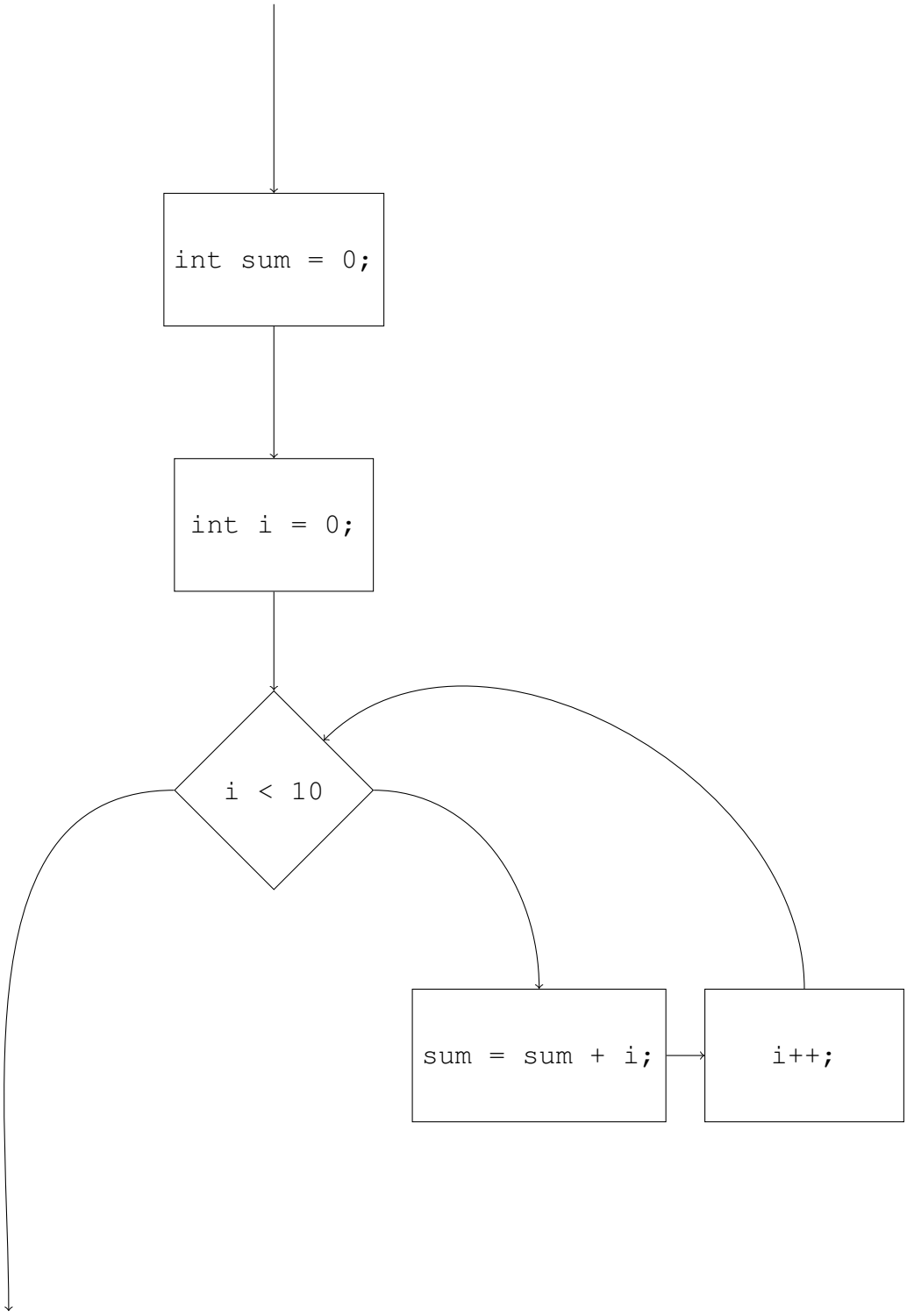
Marking scheme: 1 mark for the observation that `1 / 2` equals 0. Only $\frac{1}{2}$ mark for the observation that the result is an integer.

2 (2 marks)

Consider the following code snippet.

```
int sum = 0;
for (int i = 0; i < 10; i++)
{
  sum = sum + i;
}
```

In the control flow diagram on the next page, fill in the boxes with code from the above code snippet.



Answer:

Marking scheme: $\frac{2}{5}$ for each correct code fragment.

3 (2 marks)

Consider the following code snippet.

```
Scanner input = new Scanner(System.in); // reads from keyboard
PrintStream output = System.out; // prints to console

int value;
do
{
  output.print("Enter an integer value: ");
  value = input.nextInt();
}
while (value < 0);
```

Give code that behaves the same but uses a `while` loop instead of a `do` loop.

Answer:

```
Scanner input = new Scanner(System.in); // reads from keyboard
PrintStream output = System.out; // prints to console

output.print("Enter an integer value: ");
int value = input.nextInt();
while (value < 0)
{
  output.print("Enter an integer value: ");
  int value = input.nextInt();
}
```

or

```
Scanner input = new Scanner(System.in); // reads from keyboard
PrintStream output = System.out; // prints to console

int value = -1;
while (value < 0)
{
  output.print("Enter an integer value: ");
  int value = input.nextInt();
}
```

Marking scheme: $\frac{1}{2}$ mark for a `while` loop, $\frac{1}{2}$ mark for a correct condition of the loop, $\frac{1}{2}$ mark for a correct body of the loop, $\frac{1}{2}$ mark for correct code before the loop.

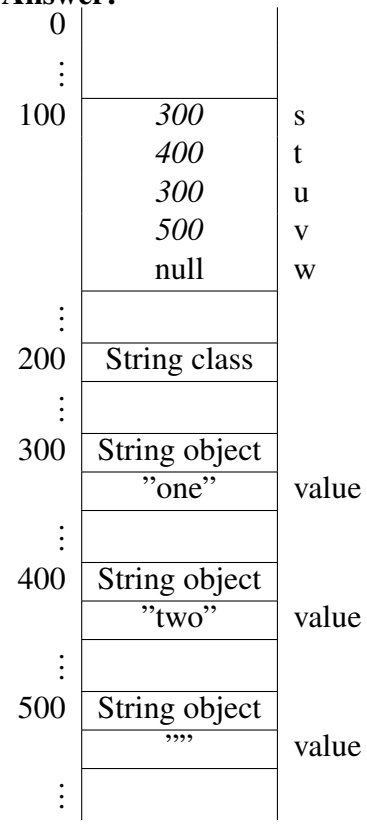
4 (2 marks)

Consider the following code snippet.

```
String s = "one";
String t = "two";
String u = "one";
String v = "";
String w = null;
```

Draw the corresponding memory diagram. Make sure that the attribute `value` and the variables `s`, `t`, `u`, `v` and `w` are reflected in your diagram. Do not include any invocation blocks.

Answer:



Marking scheme: $\frac{1}{2}$ mark if `s` and `u` refer to the same object, $\frac{1}{2}$ mark if `s` and `u` refer to different objects, $\frac{1}{2}$ mark if `s` and `v` refer to different objects, and $\frac{1}{2}$ mark if `s` does not refer to an object.

5 (2 marks)

What are the two main differences between a `List` and a `Set`?

Answer: The elements of a `List` are ordered, the elements of a `Set` are not. A `List` may contain duplicates, a `Set` does not.

Marking scheme: 1 mark for the observation that the elements of a `List` are ordered and the elements of a `Set` are not. 1 mark for the observation that a `List` may contain duplicates and a `Set` does not.

6 (2 marks)

Which interface from the collections framework (`Set`,`List` or `Map`) would be most appropriate for representing each of the collections that are described below. Not only give your choice, but also **motivate** your choice. Also, **include the type parameters** (for example, instead of `Set`, give its parameterized version, like `Set<Double>`).

(a) The creditcard numbers that have been issued by a creditcard company.

Answer: `Set<Long>` or `Set<Integer>` or `Set<String>`. Each creditcard number can represented by a `Long` or `Integer` or `String`. Since there are no duplicates, we can use a `Set` to represent this collection.

Marking scheme: $\frac{1}{4}$ mark for `Set<Long>` or `Set<Integer>` or `Set<String>`. Also $\frac{1}{4}$ mark for `List<Long>` or `List<Integer>` or `List<String>`. $\frac{1}{4}$ mark for the observation that there are no duplicates.

(b) The first names of the people who are currently customers of a creditcard company.

Answer: `List<String>`. Each first name can represented as a `String`. Since there may be duplicates, we can use a `List` to represent this collection.

Marking scheme: $\frac{1}{4}$ mark for `List<String>`. Also $\frac{1}{4}$ mark for `Set<String>`. $\frac{1}{4}$ mark for the observation that there may be duplicates.

(c) The creditcard numbers that have been issued by a creditcard company and, for each issued creditcard number, the corresponding first name of the customer.

Answer: `Map<String, String>` or `Map<Long, String>` or `Map<Integer, String>`. Each first name can represented as a `String`. Each creditcard number can be represented by a `String` or `Long` or `Integer`. To associate first names with creditcard number a `Map` can be used.

Marking scheme: $\frac{1}{4}$ mark for `Map`. $\frac{1}{4}$ mark for the observation that a `Map` can associate first names with creditcard numbers.

(d) The first names of the customers of a creditcard company and, for each customer, the creditcard numbers of the card(s) that the customer owns.

Answer: This collection be represented in many different ways. For example, `Set<Map<String, Set<String>>>`. For each customer, we introduce a `Map` from first name (`String`)

to collection of creditcard numbers (`Set<String>`). The data for all the cunstomers is collected in a `Set` (note that all `Maps` are different, that is, no duplicates). Note that `Map<String, Set<Integer>>` does not work as there may be multiple customers with the same first name.

Marking scheme: $\frac{1}{4}$ mark for a correct answer and $\frac{1}{4}$ mark for good motivation.

7 (2 marks)

Relevant fragments of the API of the `List` interface can be found at the end of this exam. Consider the following code snippet.

```
List<String> list = new ArrayList<String>();
list.add("This");
list.add("is");
list.add("an");
list.add("example");
```

(a) Write code that prints the elements of a `list` each on a separate line. For the above example, it would give rise to the following output.

```
This
is
an
example
```

Answer: This can be solved in many different ways. For example,

```
for (String word : list)
{
  System.out.println(word);
}
```

Marking scheme: $\frac{1}{2}$ mark for a loop iterating over the elements of the list. $\frac{1}{2}$ mark for extracting the element from the list.

(b) Write code that prints the elements of the `list` each on a separate line in reverse order. For the above example, it would give rise to the following output.

```
example
an
is
This
```

Answer: This can be solved in many different ways. For example,

```
for (int i = list.size() - 1; i >= 0; i--)
{
  String word = list.get(i);
  System.out.println(word);
}
```

Marking scheme: 1 mark for extracting the elements of the list in the reverse order.

8 (2 marks)

Relevant fragments of the API of the `Map` interface can be found at the end of this exam. Consider the following code snippet.

```
Map<String, String> map = new HashMap<String, String>();
map.put("This", "is");
map.put("an", "example");
```

Write code that prints the keys and elements of the `map` each on a separate line separated by a `-`. For the above example, it would give rise to the following output.

```
This - is
an - example
```

Answer: This can be solved in many different ways. For example,

```
for (String key : map.keySet())
{
  String value = map.get(key);
  System.out.println(key + " - " + value);
}
```

Marking scheme: 1 mark for using a loop to iterate over the keys. 1 mark for extracting the corresponding values from the map.

9 (2 marks)

Consider the following method.

```
private static Map<Character, Integer> getFootPrint(String word)
{
  Map<Character, Integer> footPrint = new HashMap<Character, Integer>();
  for (int i = 0; i < word.length(); i++)
  {
    char letter = word.charAt(i);
    int count = footPrint.get(letter); // this line throws a
                                       // NullPointerException
    footPrint.put(letter, count + 1);
  }
  return footPrint;
}
```

(a) **Explain** why the above method may throw a `NullPointerException`?

Answer: If the map does not contain the letter, the get method will return null, not in integer.

Marking scheme: 1 mark for the observation that get returns null or the fact that the map does not contain the letter.

(b) Describe how to change the code so that the foot print is computed correctly. Note that simply adding a `try` block and a `catch` block will not solve the problem.

Answer: We can replace the problematic line with, for example,

```
int count;
if (footPrint.get(letter) == null)
{
  count = 1;
}
else
{
  count = footPrint.get(letter) + 1;
}
```

Marking scheme: $\frac{1}{2}$ mark for checking whether the letter is in the map, $\frac{1}{2}$ mark for setting the count to 1 if the letter is not in the map.

10 (2 marks)

Consider the following code snippet.

```
public static void main(String[] args)
{
  Scanner input = new Scanner(System.in); // reads from keyboard
  PrintStream output = System.out; // prints to console

  output.print("Enter a positive integer: ");
  int size = input.nextInt();
  List<Double> list = new LinkedList<Double>();
  for (int i = 0; i < size; i++)
  {
    list.add(Math.random());
  }
  double sum = 0;
  for (int i = 0; i < size; i++)
  {
    sum = sum + list.get(i);
  }
  double average = sum / size;
  output.println("Average: " + average);
}
```

In this `main` method, the following methods may throw exceptions.

method	exception type	condition
`nextInt`	`InputMismatchException`	if the next token is not an integer
`get`	`IndexOutOfBoundsException`	if the index is out of range

Both `InputMismatchException` and `IndexOutOfBoundsException` are `RuntimeException`s.

(a) Do you have to catch or declare (to be thrown) these exceptions?

Answer: No.

Marking scheme: $\frac{2}{3}$ for the correct answer.

(b) Describe how to declare (to be thrown) these exceptions.

Answer: Add

```
throws InputMismatchException, IndexOutOfBoundsException
```

to the end of the header of the method.

Marking scheme: $\frac{1}{3}$ for throws, $\frac{1}{3}$ for the exception classes.

(c) Describe how these exceptions can be caught.

Answer: This can be done in many different ways. A simple solution is

```
try
{
  output.print("Enter a positive integer: ");
  int size = input.nextInt();
  List<Double> list = new LinkedList<Double>();
  for (int i = 0; i < size; i++)
  {
    list.add(Math.random());
  }
  double sum = 0;
  for (int i = 0; i < size; i++)
  {
    sum = sum + list.get(i);
  }
  double average = sum / size;
  output.println("Average: " + average);
}
catch (InputMismatchException e)
{
  System.out.println("Enter an integer.");
}
catch (IndexOutOfBoundsException e)
{
  // will never happen
}
```

Marking scheme: $\frac{1}{3}$ for the try block, $\frac{1}{3}$ for the catch blocks.