Name ______________________________

CISC 3115 – Modern Programming Techniques
Spring 2019
Exam 1

Part I. Place your answer in the box to the left of each question. (10 points)

B

1. In general, when developing a class in Java:
- primitive instance variables are public; class instance variable are private
- instance variables are private; methods are public
- instance variables are public; methods are private
- constructors are private; other methods are public

D

2. The keyword `this` may be used in a constructor to
- specify that there is no receiver
- indicate that it's a default constructor
- do nothing; `this` may not appear in a constructor
- call/invoke another constructor of the same class

A

3. Default constructors are called that because
- they have no parameters
- you *never* have to write them
- they automatically (i.e., without programmer assistance) supply default values to the instance variables
- *a* and *b*

C

4. An example of composition is
- having two `Name` constructors
- providing a `toString` method for each class
- having `Name` and `PhoneNumber` instance variables in the `PhonebookEntry` class
- adding a `main` method to a class

B

5. Which of the following declares and creates the phonebook array in version 2 of the project?
- `Phonebook phonebook = new PhoneBook();`
- `PhonebookEntry [] phonebook = new PhonebookEntry[100];`
- `PhonebookEntry phonebook = new Phonebook[100];`
- `PhonebookEntry [] phonebook = new PhonebookEntry(100);`

A

6. The keyword `static` is used in the declaration of a variable of the class to

- indicate that there is only one copy of the variable for the entire class
- indicate that the variable is *static* and thus cannot be changed
- indicates that the value of the variable should be initialized to 0
- specify that the variable is a primitive

B

7. `static` methods are always invoked from outside the class

- using just the method name
- using the class name instead of a receiver
- static methods are private and can't be called from outside the class
- just like any other method—with a receiver that is a variable of the class

C

8. The *handling* (i.e., processing of the exception) aspect of exception-handling is performed by the

- `throws` clause in the method header
- `throw` statement
- `try/catch` block
- All of the above

D

9. *Throwing* an exception involves

- specifying the type of exception in the `throws` clause of the method header
- printing an error message and exiting the program
- printing an error message and returning to the caller
- creating an exception object with an appropriate message and using it in a `throw` statement

B

10. Which of the following is the proper signature for the `equals` method of the `PhoneNumber` class from Lab 3.2?

- `public boolean equals()`
- `public boolean equals(PhoneNumber p)`
- `public boolean equals(PhoneNumber p1, PhoneNumber p2)`
- `public boolean equals(PhoneNumber this, PhoneNumber other)`

Part II. Place answer all questions.

11. a. **(10 points)** Define a `Counter` class with the following state and behavior:

- An integer instance variable to maintain the value of the counter
- Methods `up` and `down` that increase and decrease the value of the counter by 1.
- A method `getVal` that returns the current value of the counter
- A `reset` method that results in the Counter's value being set to back 0
- A `toString` method that returns a string of the form: "A counter with the value *val*"

```
class Counter {
     public void up() {val++;}
     public void down() {val--;}

     public void reset() {val = 0;}

     public int getVal() {return val;}

     public String toString() {return "A counter with the value  " + val;}

     private int val;
}
```

b. (**5 points**) Write a fragment of code that illustrates the use of your `Counter` class: declaring and creating a `Counter` object, using a for loop to increment the counter 10 times, printing out the counter, and then a loop that decrements the counter back down to 0.

```
Counter c = new Counter();

for (int i = 1; i <= 10; i++)
     c.up();

System.out.println(c);

while (c.getVal() != 0)
     c.down();
```

12. a. (**5 points**) Given a `Name` class with (the usual) `first` and `last` instance variables (of type `String`), write the (static) method `read` (presented in class) that accepts a `Scanner` variable, and creates and returns a new `Name` object by reading data from the `Scanner` and using it to create (and return) the new `Name` object

```
public static Name read(Scanner scanner) {
     if (!scanner.hasNext()) return null;
     String last = scanner.next();
     String first = scanner.next();
     return new Name(last, first);
}
```

Makes no difference as to which order you read last and first name

b. (**5 points**) Write a fragment of code that declares an array of size 100, and reads in a series of `Name` objects (using the above `read` method) into the array. Make sure you include logic to ensure you don't overflow the array (throw an exception if the array capacity is exceeded). Assume a `Scanner` variable named `scanner` has been declared and assigned a `Scanner` object created and opened to the proper file.

```
final int CAPACITY = 100;
Name [] names = new Name[CAPACITY];
int size = 0;

Name name = Name.read(scanner);
while (name != null) {
     if (size >= CAPACITY) {
          throw new Exception("Increase the size of your array");
     names[size++] = name;
     name = Name.read(scanner);
}
```

13. Given the following `Student` class that contains a `Name` instance variable (`Name` having the usual `first/last` state and get methods):

```
1  class Student {
2        Student(Name name) {
3              id = nextId;
4              nextId++;
5              this.name = name;
6        }

7        String getFirstName() {return name.getFirst();}

8        int id;
9        Name name;
10       static int nextId = 1001;
11 }
```

a. (**5 points**) Why is the `getFirstName` method of `Student` (line 7) called a *delegation* method?

Because it does its job by calling the `getFirst` method of the `name` instance variable; i.e., it delegates its work to the contained object.

b. (**5 points**) Why is `nextId` static? What is the purpose of the `nextId` variable? What is happening in lines `3` and `4`?

`nextId` is used to assign sequential id's to `Student` objects as they are created. We want this number to be unique so we make it static (i.e., a class variable) so there is only one copy of it in the class. In line3 3 and 4 we assign the current value of `nextId` to the `id` instance variable (of which there is one per `Student` instance/object, and then update `nextId` so the next created object gets a different (unique and sequential) value in its `id`.

14. (**5 points**) What is happening in the following code fragment, i.e., what does the logic of the loop do, and what happens once you get to the … after the loop?

```
Scanner scanner;
while (!done) {
    try {
             System.out.print("filename? ");
              String filename = keyboard.next();
              Scanner = new Scanner(new File(filename);
             done = true;
     } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }
}
…
```

The code prompts the user for a file name and attempts to open a Scanner object with the corresponding file. If the open is successful, the loop exits (since the value of done is set in the statement after the open; if the open fails an exception is thrown (the assignment of true to done is therefore skipped) and the process is repeated.

Upon exiting the loop we are guaranteed to have a properly opened Scanner object