CSC 210. FINAL EXAM FALL 2020

1. Section Date and Time:
- CSC 210.03 Wednesday, December 16, 12:35 – 2:35 PM
2. Final Exam (1 exam): 150 points

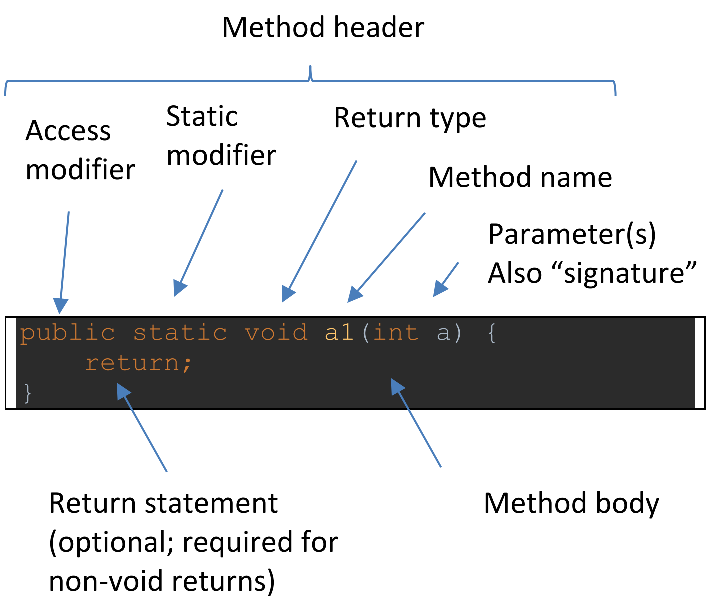
Full Name	**SFSU ID**
Kullathon Sitthisarnwattanachai	921425216

3. Please review all related material Lecture notes, slides, midterm practices and exam, in-class practice, sample programs.
4. Show your work typed. Your writing must be clear and readable to the grader(s).

HONOR CODE: - Please follow CS Department's policies: https://cs.sfsu.edu/cheating-and-plagiarism-policy

PART A – 35 Points

A.1 - 7 Points
Please write a complete **method declaration** and name each component.


Method header
Access modifier
Static modifier
Return type
Method name
Parameter(s) Also "signature"
Return statement (optional; required for non-void returns)
Method body

```java
public static void a1(int a) {
    return;
}
```

A.2 - 7 Points
Please write code to declare and initialize a **1D array** then write a **do-while loop** to display the content of the array.

```java
public class A2 {
    public static void main(String[] args) {
        int[] a2 = {1, 2, 3};
        int i = 0;
        do {
            System.out.println(a2[i]);
            i++;
        } while (i < a2.length);
    }
}
```

A.3 - 7 Points
Please write code to declare and initialize a **2D array** then write a **for loop** to display the content of the array.

```java
public class A3 {
    public static void main(String[] args) {
        int[][] a3 = {{1, 2}, {2, 2}, {3, 2}};
        for (int i = 0; i < a3.length; i++) {
            for (int j = 0; j < a3[i].length;
j++) {
                System.out.printf("%d ",
a3[i][j]);
            }
            System.out.println();
        }
    }
}
```

A.4 - 7 Points
Please **list** and **explain** the general components of a Java class.

1. **Class modifier**
   e.g., public. Defines the access type for the class. Public classes are accessible by all classes. Classes without the public keyword will only be accessible within its own package.
2. **Class name**
   The name to use when identifying the class.
3. **Fields**
   Variables inside of a class used to keep track of the "states" of the class.
4. **Getters and setters**
   Methods used to get and set the fields for the class, respectively. Fields may be kept private to prevent unnecessary access from outside the class. Setters can be used to sanitize input for the class and getters can be used to define how the fields are represented.
5. **Constructors**
   Special methods used to create an instance of the class ("construct"). The method must have the same name as the class. Implicitly returns an instance of the class with the fields initialized, if any.
6. **Methods**
   a. **Instance methods**
   Methods belonging to the class that can only be accessed via the dot notation on an instance of class. These are methods that either modify or depends on

the values of the fields. These classes do not have the "static" modifier in its header.

b. Static methods

Contra to instance methods, static methods can be accessed directly via the dot notation on the class itself. It can also be accessed via an instance of that class; however, its behavior will not differ. Static methods are methods that do not depend on the state of the fields of the class. They have the "static" keyword in its header and cannot invoke other instance methods within itself.

A.5 - 7 Points
Please explain in detail the differences between a **static** and an **instance** variable.

A **static variable** is a variable that is consistent throughout all instances of the class. The variable is said to belong to the class (instead of an instance of the class). The variables can be accessed (if not private) via the dot notation on the class itself and can also on an instance of that class. If the variable can be accessed outside the scope of the class itself (not private) and is not final, then the variable can be manipulated outside the class.

An **instance variable** is a variable that belongs to a particular instance of the class (instead of the class itself). An instance variable is used to keep track of different states of the instance of the class and therefore can only be accessed via an instance and not the class itself. The value of an instance variable may vary between different instances. On the contrary, the value of the static variable would be consistent throughout different instances of a class (since it belongs to the class itself) at a particular point.

PART B – 100 Points

B.1 - 15 Points
Please code a complete **if-else statement** then convert it into a **switch statement**. Please have at least 4 cases (including the default case).

if-else statement:

```
if (i == 1) {
    System.out.println("A");
} else if (i == 2) {
    System.out.println("B");
} else if (i == 3) {
    System.out.println("C");
} else if (i == 4) {
    System.out.println("D");
}
```

switch statement:

```
switch (i) {
    case 1:
        System.out.println("A");
        break;
    case 2:
        System.out.println("B");
        break;
    case 3:
        System.out.println("C");
        break;
    case 4:
        System.out.println("D");
        break;
    default:
        break;
}
```

B.2 - 15 Points
Please code **a complete Java program**: **`HolidayStudio`**
- Your program prompts users to enter their favorite language.
- Then the program prints "Merry Xmas and Happy New Year!" in that language.
- It is OK to assume that users will always choose your favorite language.
- This program must have at least 2 methods. (1 of them is the **`main`** method.)
- A sample run of the program (think Google Translate):

```
Enter favorite language: Vietnamese

Chúc Giáng Sinh an lành và Năm Mới hạnh phúc!

Enter favorite language: Indonesian

Selamat natal dan tahun baru!

Enter favorite language: English

Merry Xmas and Happy New Year!

Enter favorite language: exit

Goodbye!
```

```
import java.util.Scanner;

public class HolidayStudio {

    public static final Scanner scan = new
Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            String lang = promptLanguage();
            if (lang.equals("exit")) {

System.out.println("Goodbye!");
                break;
            }

System.out.println(getGreetings(lang));
        }
    }

    public static String promptLanguage() {
        System.out.print("Enter favorite
language: ");
        return scan.nextLine();
    }

    public static String getGreetings(String
lang) {
        switch (lang.toLowerCase()) {
            case "vietnamese":
                return "Chúc Giáng Sinh an
lành và Năm Mới hạnh phúc!";
            case "indonesian":
                return "Selamat natal dan
tahun baru!";
            case "english":
                return "Merry Xmas and Happy
New Year!";
            default:
                return "";
        }
    }
}
```

B.3-TO-B.6 – B.3 10 points
Please think of a real-life entity which you can write a Java class to represent.

What is that entity?

An SFSU student.

Why do you think it is suitable to be a Java class?

They contain properties can be turned into a "blueprint." For example, a student's name, DOB, address, ID number, etc. can be stored in to fields. Actions such as adding classes can be turned into methods. They can also be subclassed for different types of students, such as undergrad, postgrad, etc.

Please code the class.

```
public class SFSUStudent {
    private String firstName;
    private String lastName;
    private String preferredName;
    private int age;
    private int ID;
    private String[] classes;

    public SFSUStudent(String firstName,
String lastName, String preferredName, int
age, int ID, String[] classes) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.preferredName = preferredName;
        this.age = age;
        this.ID = ID;
        this.classes = classes;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName)
{
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getPreferredName() {
        return preferredName;
    }

    public void setPreferredName(String
preferredName) {
        this.preferredName = preferredName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String[] getClasses() {
        return classes;
    }

    public void setClasses(String[] classes) {
        this.classes = classes;
    }

    public void addClass(String newClass) {
        String[] newClasses = new
String[classes.length + 1];
        for (int i = 0; i < classes.length;
i++) {
            newClasses[i] = classes[i];
        }
        newClasses[classes.length + 1] =
newClass;
        classes = newClasses;
    }
}
```

Please code a data field for your class to keep track of the number of objects created.

Below is the addition of the field used to keep track of the number of student and the sole constructor that is modified to increment each time the class has been instantiated. A getter is also provided to access the field. Note the exclusion of a setter for this field is intentional. The rest have been left out for brevity as it remains the same.

```
private static int numberOfStudents;

public SFSUStudent(String firstName, String
lastName, String preferredName, int age, int
ID, String[] classes) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.preferredName = preferredName;
    this.age = age;
    this.ID = ID;
    this.classes = classes;
    numberOfStudents++;
}

public int getNumberOfStudents() {
    return numberOfStudents;
}
```

Then explain in detail why your code should work properly, including how the data field should be used, and whether it should be **static, public** or **private**.

All the data fields are private to prevent unnecessary access to the field and to prevent any unintentional changes.

To access the fields of the instance, one must use the getters to do so. The setters have been included, should any further sanitation be implemented to control the input.

All the fields, except for one, are not static since they should only be concerned with a particular instance (in this case, a student).

The only field that is static is the variable used to keep track of the number of students in the school. This was added as part of this question. The field is static for the count to be consistent throughout all instances of the class since the count belongs to the class itself. This field is also private to prevent any unnecessary access. The field is only incremented in the sole constructor of the class and does not have a setter to ensure that the count is only updated when a new instance of a student have been created.

Aside from the getters and setters, one method have been implemented to add classes to the student. It simply clones the old list of class and effectively appends the new class on to the

B.4 - 10 Points

old one. The old array of class is then overridden by the new one.

B.5 - 10 Points
Write a no-argument constructor and a three-argument constructor for your class.

```
public SFSUStudent(String firstName, String
lastName, String preferredName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.preferredName = preferredName;
    numberOfStudents++;
}

public SFSUStudent() {
    numberOfStudents++;
}
```

Code instructions to create 2 objects using both constructors above.

```
public static void main(String[] args) {
    SFSUStudent s1 = new SFSUStudent();
    SFSUStudent s2 = new
SFSUStudent("Kullathon",
"Sitthisarnwattanachai", "Mos");
}
```

B.6 - 10 Points
Please code a **static** and a **private** variables for your class.

The field added for B4 is a private and static variable.

```
private static int numberOfStudents;
```

The rest of the fields in the class are private.

```
private String firstName;
private String lastName;
private String preferredName;
private int age;
private int ID;
private String[] classes;
```

Choose 1 variable and write 2 statements to show 2 ways to access the variable from **`main`** method. Mark the preferred way.

If the main method was inside the class itself. For normal access outside of the class, the getter needs to be used and is preferred if the variable was public. If the variable is accessed internally, the dot notation referring to the variable itself is preferred.

```
public static void main(String[] args) {
    SFSUStudent s2 = new
SFSUStudent("Kullathon",
"Sitthisarnwattanachai", "Mos");
    // using getter is preferred (and
needed) outside class

System.out.println(s2.getFirstName());
    System.out.println(s2.firstName);
}
```

B.7 - 10 Points
What are the 3 pillars of Object-Oriented Programming? Please explain each of them in your own word.

1. **Abstraction**
   Abstractions allows the code to be used without understanding the inner-workings of the implementations.
2. **Encapsulation**
   Containing the variables and methods that are related to one another into a single class. This allows the access for the methods and variables to be defined more precisely, and thus prevent unnecessary access to them.
3. **Inheritance**
   "Inheriting" a class from another. This allows new classes to be created from existing ones. It allows fields and methods from an existing class to be used, and they can be "overridden" to define new behaviors specific to the subclass.

B.8 - 10 Points
Please code a new and encapsulated data field to add to your class.

Here is a new field called gender and its setter and getter.

```
private String gender;

public String getGender() {
    return gender;
}

public void setGender(String gender) {
    this.gender = gender;
}
```

Please code what you need in order to be able to update the data field using an object created in **main**.

Invoking the setter to set the gender field for the instance.

```
public static void main(String[] args) {
    SFSUStudent s1 = new SFSUStudent();
    SFSUStudent s2 = new
SFSUStudent("Kullathon",
"Sitthisarnwattanachai", "Mos");
     s2.setGender("M");
}
```

B.8 - 10 Points

State whether following statements of naming conventions are true or false.

1. All variable names should start with an uppercase letter. (T/F)? **False.**
2. Method names should be lower-cased.(T/F)? **False, they are lowerCamelCase.**
3. If a variable name contains more than one word, capitalize the first letter of each word except first.(T/F)? **True, this is lowerCamelCase.**
4. Capitalize the first letter in each word of class name.(T/F)? **True, including the first. This is UpperCamelCase.**

PART C – 15 Points

C.1 - 5 Points
Give a your own example of method overloading and method overriding

```
public class C1 {
    // method overloading
    public static void meth1(String a) {
        System.out.println(a);
    }

    public static void meth1(int a) {
        System.out.println(a * 5);
    }
}
```

```
// method overriding
class C2 extends C1 {
    @Override
    public static void meth1(int a) {
        System.out.println(a * 2);
    }
}
```

C.2 - 5 Points
Give a real-life example of an **Inheritance** relationship

A dog of a particular breed, say chihuahua, inherits a lot of the features from any particular dog, from which inherits the features from animals. In a java class, the Chihuahua class can inherit from a class called Dog, which can also inherit from the class called Animal.

What is the keyword used in the Child Class header?

"extends"

C.3 - 5 Points Why you would you use a try-catch statement ?

To catch any edge-cases where an error may arise. For example, a user might input something that you might not expect. If you use a scanner to scan in an integer but a user enters a String (that cannot be cast into an int), then the program will abruptly stop due to an exception being thrown. This can be prevented by using a try-catch statement to handle this particular case. A message can be displayed to the user and the program can prompt the user to enter the input again. In other words, it can be used to handle errors more gracefully.