Rafael Antonio López García
Programación Orientada a Objetos

Preguntas de revisión

- Preguntas en azul.
- Respuestas en verde.

1. The code between a pair of curly braces in a method is a:
   a. function
   b. block
   c. brick
   d. sector
2. When a block exists within another block, the blocks are:
   a. structured
   b. nested
   c. sheltered
   d. illegal
3. The portion of a program within which you can reference a variable is the variable's:
   a. range
   b. space
   c. domain
   d. scope
4. You can declare variables with the same name multiple times:
   a. within a statement
   b. within a block
   c. within a method
   d. You never can declare multiple variables with the same name.
5. If you declare a variable as an instance variable within a class, and you declare and use the same variable name within a method of the class, then within the method:
   a. the variable used inside the method takes precedence.
   b. the class instance variable takes precedence.
   c. the two variables refer to a single memory address.
   d. an error will occur.

6. A method variable ___ a class variable with the same name.
   a. acquiesces to
   b. destroys
   c. overrides
   d. alters

7. Non ambiguous, overloaded methods must have the same:
   a. name
   b. number of parameters
   c. parameter names
   d. types of parameters

8. If a method is written to receive a double parameter, and you pass an integer to the method, then the method will:
   a. work correctly; the integer will be promoted to a double.
   b. work correctly; the integer will remain an integer.
   c. execute, but any output will be incorrect.
   d. not work; an error message will be issued.

9. A constructor ___ parameters.
   a. can receive.
   b. cannot receive.
   c. must receive.
   d. can receive a maximum of 10.

10. A constructor ___ overloaded:
    a. can be.
    b. cannot be.
    c. must be.
    d. is always automatically.

11. Usually, you want each instantiation of a class to have its own copy of:
    a. the data fields.
    b. the class methods.

c. both above.

d. none of the above.

12. If you create a class that contains one method and instantiate two objects, you usually store for use with the objects:

a. one copy of the method

b. two copies of the method

c. two different methods containing two different "this" references.

d. data only (the methods are not stored)

13. The "this" reference:

a. can be used implicitly.

b. must be used implicitly.

c. must not be used implicitly.

d. must not be used.

14. Methods that you reference with individual objects are:

a. private

b. public

c. static

d. non-static

15. Variables that are shared by every instantiation of a class are:

a. class variables

b. private variables

c. public variables

d. illegal

16. The keyword final used with a variable declaration indicates:

a. the end of the program.

b. a static field.

c. a symbolic constant.

d. that no more variables will be declared in the program.

17. Java classes are stored in a folder or:

a. packet
b. package
c. bundle
d. gaggle

18. Which of the following statements determines the square root of a number and assigns it to the variable ‘s’?

a. s = sqrt(number);
b. s = Math.sqrt(number);
c. number = sqrt(s);
d. number = Math.sqrt(s);

19. A GregorianCalendar object can be created with one of seven constructors. This means that the constructors:

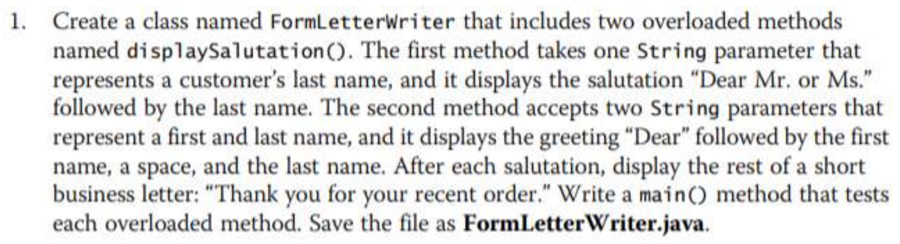
a. override each other.
b. are ambiguous.
c. are overloaded.
d. all of the above.

20. The GregorianCalendar class get() method always returns a(n):

a. day of the week.
b. date
c. integer
d. GregorianCalendar object.

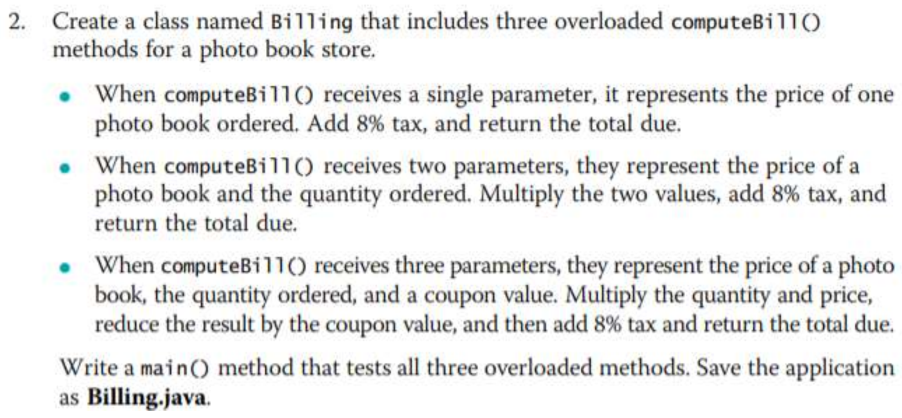
Ejercicios de programación

- Enlaces a los archivos en azul.



1. Create a class named `FormLetterWriter` that includes two overloaded methods named `displaySalutation()`. The first method takes one `String` parameter that represents a customer's last name, and it displays the salutation "Dear Mr. or Ms." followed by the last name. The second method accepts two `String` parameters that represent a first and last name, and it displays the greeting "Dear" followed by the first name, a space, and the last name. After each salutation, display the rest of a short business letter: "Thank you for your recent order." Write a `main()` method that tests each overloaded method. Save the file as **FormLetterWriter.java**.

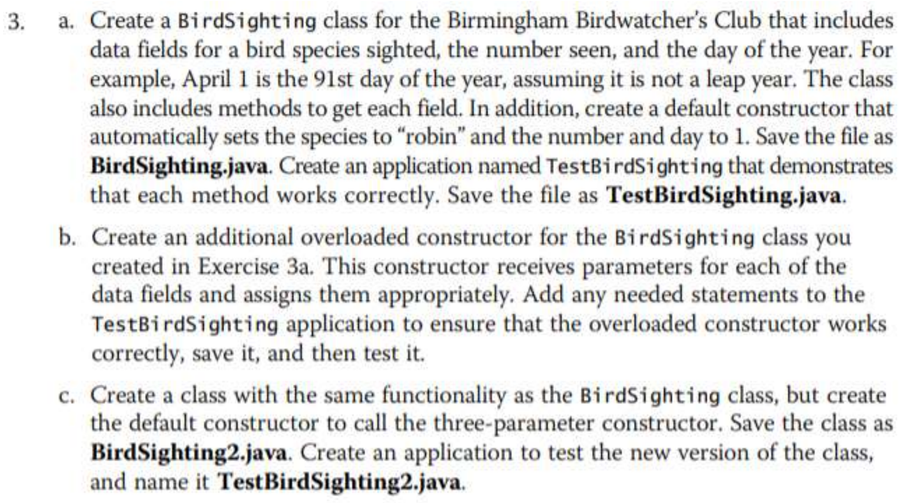
Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio1



2. Create a class named `Billing` that includes three overloaded `computeBill()` methods for a photo book store.
   - When `computeBill()` receives a single parameter, it represents the price of one photo book ordered. Add 8% tax, and return the total due.
   - When `computeBill()` receives two parameters, they represent the price of a photo book and the quantity ordered. Multiply the two values, add 8% tax, and return the total due.
   - When `computeBill()` receives three parameters, they represent the price of a photo book, the quantity ordered, and a coupon value. Multiply the quantity and price, reduce the result by the coupon value, and then add 8% tax and return the total due.

   Write a `main()` method that tests all three overloaded methods. Save the application as **Billing.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio2



3. a. Create a `BirdSighting` class for the Birmingham Birdwatcher's Club that includes data fields for a bird species sighted, the number seen, and the day of the year. For example, April 1 is the 91st day of the year, assuming it is not a leap year. The class also includes methods to get each field. In addition, create a default constructor that automatically sets the species to "robin" and the number and day to 1. Save the file as **BirdSighting.java**. Create an application named `TestBirdSighting` that demonstrates that each method works correctly. Save the file as **TestBirdSighting.java**.

   b. Create an additional overloaded constructor for the `BirdSighting` class you created in Exercise 3a. This constructor receives parameters for each of the data fields and assigns them appropriately. Add any needed statements to the `TestBirdSighting` application to ensure that the overloaded constructor works correctly, save it, and then test it.

   c. Create a class with the same functionality as the `BirdSighting` class, but create the default constructor to call the three-parameter constructor. Save the class as **BirdSighting2.java**. Create an application to test the new version of the class, and name it **TestBirdSighting2.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio3

4. a. Create a class named `BloodData` that includes fields that hold a blood type (the four blood types are *O*, *A*, *B*, and *AB*) and an Rh factor (the factors are + and –). Create a default constructor that sets the fields to "O" and "+", and an overloaded constructor that requires values for both fields. Include get and set methods for each field. Save this file as **BloodData.java**. Create an application named `TestBloodData` that demonstrates that each method works correctly. Save the application as **TestBloodData.java**.

   b. Create a class named `Patient` that includes an ID number, age, and `BloodData`. Provide a default constructor that sets the ID number to "0", the age to 0, and the `BloodData` to "O" and "+". Create an overloaded constructor that provides values for each field. Also provide get methods for each field. Save the file as **Patient.java**. Create an application named `TestPatient` that demonstrates that each method works correctly, and save it as **TestPatient.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio4

5. a. Create a class for the Tip Top Bakery named `Bread` with data fields for bread type (such as "rye") and calories per slice. Include a constructor that takes parameters for each field, and include get methods that return the values of the fields. Also include a `public final static String` named `MOTTO` and initialize it to *The staff of life*. Write an application named `TestBread` to instantiate three `Bread` objects with different values, and then display all the data, including the motto, for each object. Save both the **Bread.java** and **TestBread.java** files.

   b. Create a class named `SandwichFilling`. Include a field for the filling type (such as "egg salad") and another for the calories in a serving. Include a constructor that takes parameters for each field, and include get methods that return the values of the fields. Write an application named `TestSandwichFilling` to instantiate three `SandwichFilling` objects with different values, and then display all the data for each object. Save both the **SandwichFilling.java** and **TestSandwichFilling.java** files.

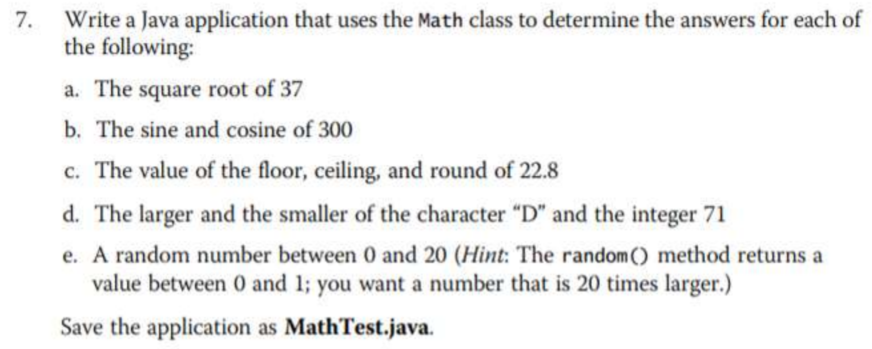
   c. Create a class named `Sandwich`. Include a `Bread` field and a `SandwichFilling` field. Include a constructor that takes parameters for each field needed in the two objects and assigns them to each object's constructor. Write an application named `TestSandwich` to instantiate three `Sandwich` objects with different values, and then display all the data for each object, including the total calories in a `Sandwich`, assuming that each `Sandwich` is made using two slices of `Bread`. Save both the **Sandwich.java** and **TestSandwich.java** files.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio5

6. a. Create a class named `Circle` with fields named `radius`, `diameter`, and `area`. Include a constructor that sets the radius to 1 and calculates the other two values. Also include methods named `setRadius()`and `getRadius()`. The `setRadius()` method not only sets the radius but also calculates the other two values. (The diameter of a circle is twice the radius, and the area of a circle is *pi* multiplied by the square of the radius. Use the `Math` class `PI` constant for this calculation.) Save the class as **Circle.java**.

   b. Create a class named `TestCircle` whose `main()` method declares several `Circle` objects. Using the `setRadius()` method, assign one `Circle` a small radius value, and assign another a larger radius value. Do not assign a value to the radius of the third circle; instead, retain the value assigned at construction. Display all the values for all the `Circle` objects. Save the application as **TestCircle.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio6

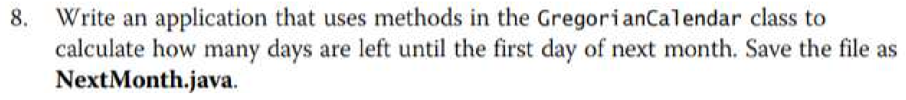


7. Write a Java application that uses the `Math` class to determine the answers for each of the following:

   a. The square root of 37
   b. The sine and cosine of 300
   c. The value of the floor, ceiling, and round of 22.8
   d. The larger and the smaller of the character "D" and the integer 71
   e. A random number between 0 and 20 (*Hint*: The `random()` method returns a value between 0 and 1; you want a number that is 20 times larger.)

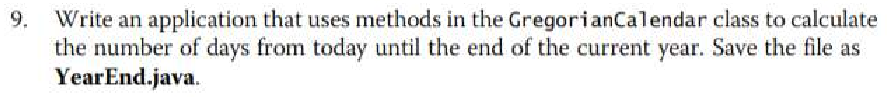
   Save the application as **MathTest.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio7



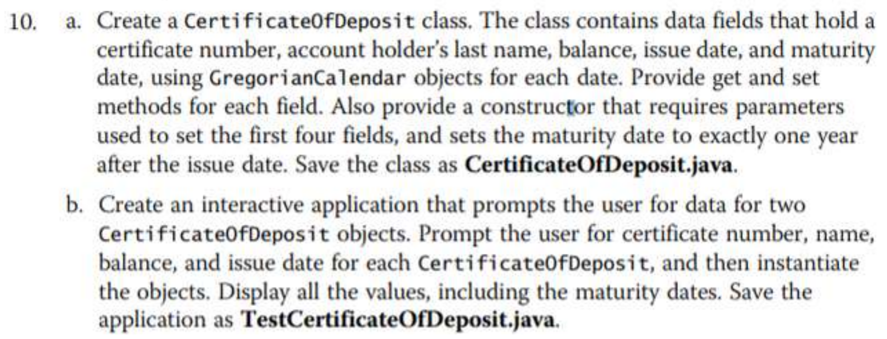
8. Write an application that uses methods in the `GregorianCalendar` class to calculate how many days are left until the first day of next month. Save the file as **NextMonth.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio8



9. Write an application that uses methods in the `GregorianCalendar` class to calculate the number of days from today until the end of the current year. Save the file as **YearEnd.java**.

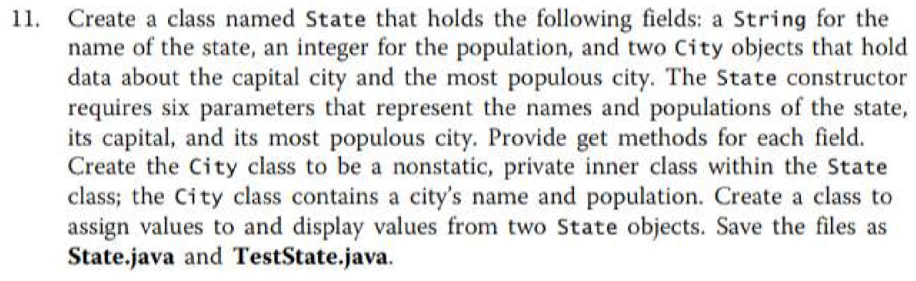
Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio9



10. a. Create a `CertificateOfDeposit` class. The class contains data fields that hold a certificate number, account holder's last name, balance, issue date, and maturity date, using `GregorianCalendar` objects for each date. Provide get and set methods for each field. Also provide a constructor that requires parameters used to set the first four fields, and sets the maturity date to exactly one year after the issue date. Save the class as **CertificateOfDeposit.java**.

    b. Create an interactive application that prompts the user for data for two `CertificateOfDeposit` objects. Prompt the user for certificate number, name, balance, and issue date for each `CertificateOfDeposit`, and then instantiate the objects. Display all the values, including the maturity dates. Save the application as **TestCertificateOfDeposit.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio10



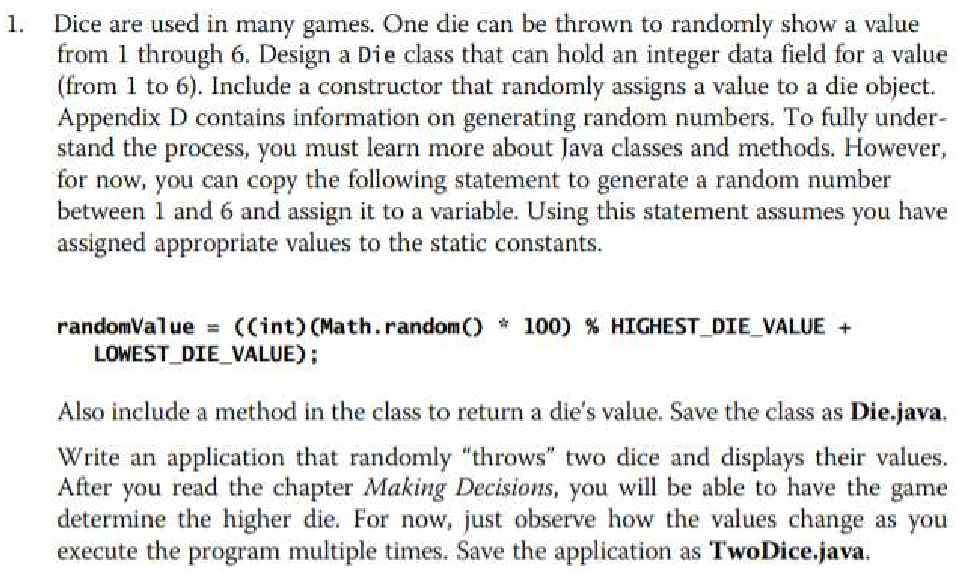
11. Create a class named `State` that holds the following fields: a `String` for the name of the state, an integer for the population, and two `City` objects that hold data about the capital city and the most populous city. The `State` constructor requires six parameters that represent the names and populations of the state, its capital, and its most populous city. Provide get methods for each field. Create the `City` class to be a nonstatic, private inner class within the `State` class; the `City` class contains a city's name and population. Create a class to assign values to and display values from two `State` objects. Save the files as **State.java** and **TestState.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/ejercicio11

Rafael Antonio López García
Programación Orientada a Objetos

Game Zone

- Enlaces a los archivos en azul.



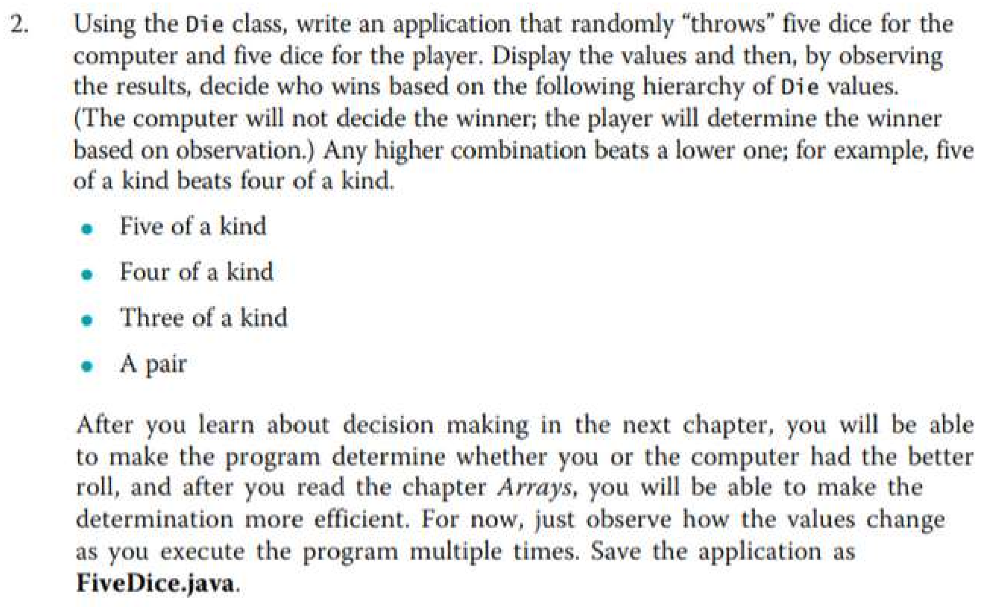
1. Dice are used in many games. One die can be thrown to randomly show a value from 1 through 6. Design a `Die` class that can hold an integer data field for a value (from 1 to 6). Include a constructor that randomly assigns a value to a die object. Appendix D contains information on generating random numbers. To fully understand the process, you must learn more about Java classes and methods. However, for now, you can copy the following statement to generate a random number between 1 and 6 and assign it to a variable. Using this statement assumes you have assigned appropriate values to the static constants.

```
randomValue = ((int)(Math.random() * 100) % HIGHEST_DIE_VALUE +
   LOWEST_DIE_VALUE);
```

Also include a method in the class to return a die's value. Save the class as **Die.java**.

Write an application that randomly "throws" two dice and displays their values. After you read the chapter *Making Decisions*, you will be able to have the game determine the higher die. For now, just observe how the values change as you execute the program multiple times. Save the application as **TwoDice.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/gameZone1



2. Using the `Die` class, write an application that randomly "throws" five dice for the computer and five dice for the player. Display the values and then, by observing the results, decide who wins based on the following hierarchy of `Die` values. (The computer will not decide the winner; the player will determine the winner based on observation.) Any higher combination beats a lower one; for example, five of a kind beats four of a kind.
   - Five of a kind
   - Four of a kind
   - Three of a kind
   - A pair

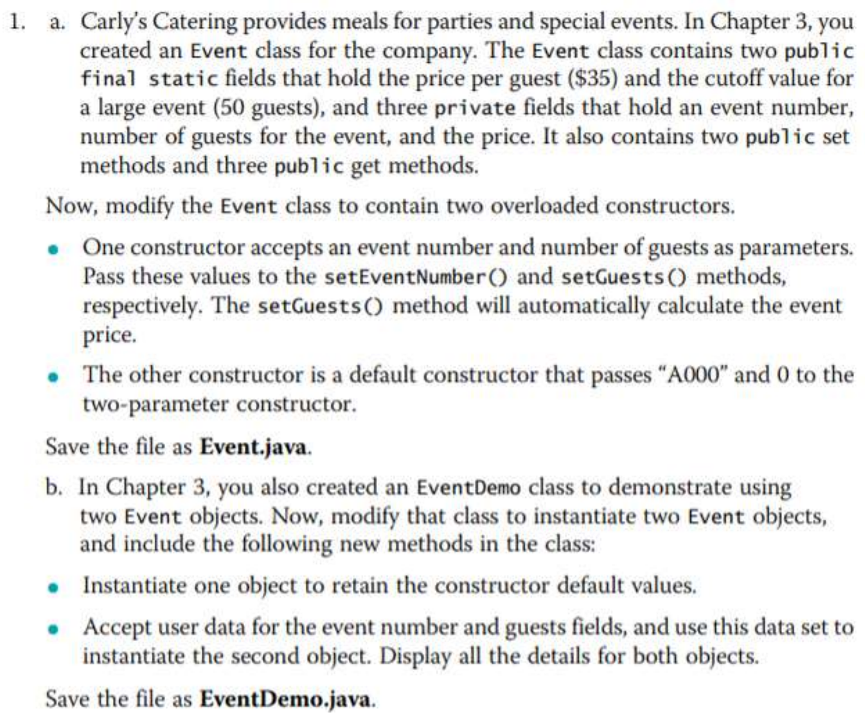
   After you learn about decision making in the next chapter, you will be able to make the program determine whether you or the computer had the better roll, and after you read the chapter *Arrays*, you will be able to make the determination more efficient. For now, just observe how the values change as you execute the program multiple times. Save the application as **FiveDice.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/gameZone2

Rafael Antonio López García
Programación Orientada a Objetos

Case Problems

- Enlaces a los archivos en azul.



1. a. Carly's Catering provides meals for parties and special events. In Chapter 3, you created an `Event` class for the company. The `Event` class contains two `public final static` fields that hold the price per guest ($35) and the cutoff value for a large event (50 guests), and three `private` fields that hold an event number, number of guests for the event, and the price. It also contains two `public` set methods and three `public` get methods.

   Now, modify the `Event` class to contain two overloaded constructors.

   - One constructor accepts an event number and number of guests as parameters. Pass these values to the `setEventNumber()` and `setGuests()` methods, respectively. The `setGuests()` method will automatically calculate the event price.
   - The other constructor is a default constructor that passes "A000" and 0 to the two-parameter constructor.

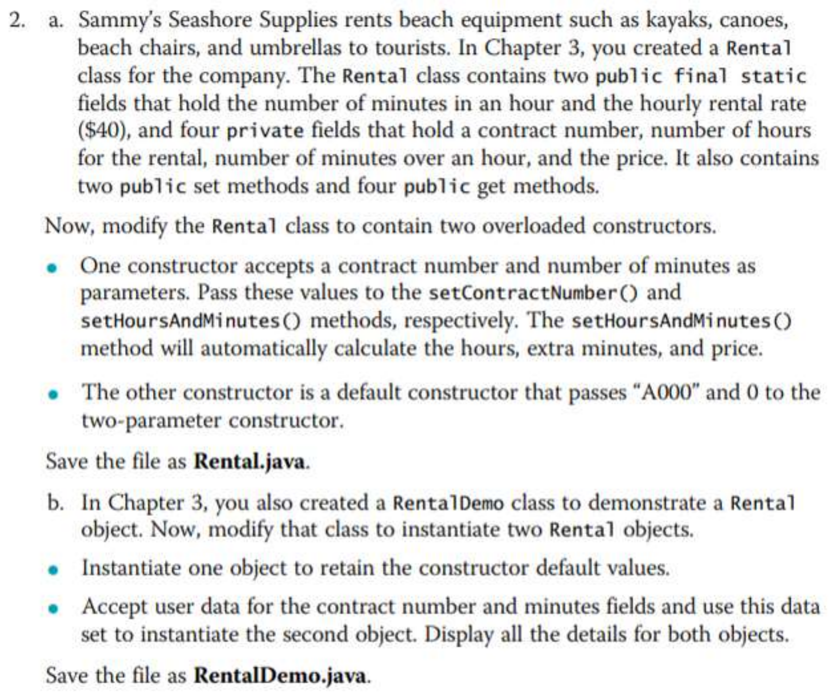
   Save the file as **Event.java**.

   b. In Chapter 3, you also created an `EventDemo` class to demonstrate using two `Event` objects. Now, modify that class to instantiate two `Event` objects, and include the following new methods in the class:

   - Instantiate one object to retain the constructor default values.
   - Accept user data for the event number and guests fields, and use this data set to instantiate the second object. Display all the details for both objects.

   Save the file as **EventDemo.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/caseProblem1



2. a. Sammy's Seashore Supplies rents beach equipment such as kayaks, canoes, beach chairs, and umbrellas to tourists. In Chapter 3, you created a `Rental` class for the company. The `Rental` class contains two `public final static` fields that hold the number of minutes in an hour and the hourly rental rate ($40), and four `private` fields that hold a contract number, number of hours for the rental, number of minutes over an hour, and the price. It also contains two `public` set methods and four `public` get methods.

   Now, modify the `Rental` class to contain two overloaded constructors.

   - One constructor accepts a contract number and number of minutes as parameters. Pass these values to the `setContractNumber()` and `setHoursAndMinutes()` methods, respectively. The `setHoursAndMinutes()` method will automatically calculate the hours, extra minutes, and price.
   - The other constructor is a default constructor that passes "A000" and 0 to the two-parameter constructor.

   Save the file as **Rental.java**.

   b. In Chapter 3, you also created a `RentalDemo` class to demonstrate a `Rental` object. Now, modify that class to instantiate two `Rental` objects.

   - Instantiate one object to retain the constructor default values.
   - Accept user data for the contract number and minutes fields and use this data set to instantiate the second object. Display all the details for both objects.

   Save the file as **RentalDemo.java**.

Enlace al archivo:
https://github.com/logralahad/POO1_Capitulo4/tree/main/chapter4_Joyce/src/caseProblem2