Practices for Lesson 10

Practices Overview

In these practices, you experiment with field access and encapsulation, and create and use overloaded constructors. A challenge practice is included here for those of you who have extra time and want to be challenged.

Practice 10-1: Implementing Encapsulation in a Class

Overview

In this practice, you create a class containing private attributes and try to access them in another class. This practice has two sections.

- Implementing encapsulation in a class
- Accessing encapsulated attributes of a class

Assumptions

This practice assumes that the following files appear in the practice folder for this lesson, `Lesson10`, and consequently also in the Practice11 project.

- DateOneTest.java
- DateTwoTest.java
- DateThreeTest.java

Implementing Encapsulation in a Class

1. Create a new project called "Practice11". Refer to the Practice 1-2, Steps 3 and 4 if you need assistance.
2. Create a new Java Class called "DateOne". Declare three member fields of type `int` named: `day`, `month`, and `year`. Give public access to all the member fields.
3. Open the DateOneTest class. In the `main` method, do the following:
   a. Create and initialize an object of type `DateOne`.
   b. Assign different numeric values to the member fields of the DateOne object.
   c. Display the value of the member fields of the DateOne object. Concatenate them into a single String with your choice of date formatting.

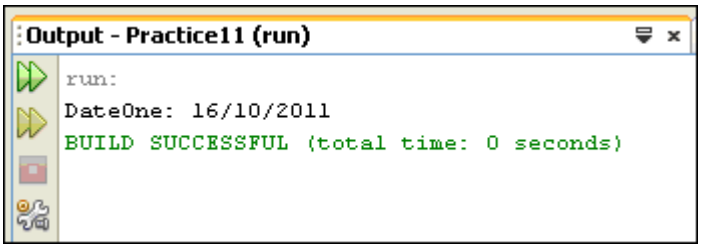
   Note: The back slash (\) character is a special character in the Java language called an "escape character". If you want to use it as part of your date format, use two back slashes together to have one of the backslashes appear in the String. Example: day + "\\" + month results in a single backslash between day and month. There are no restrictions for using a forward slash.

   Solution:

```
public static void main(String args[]){
     DateOne date = new DateOne();
     date.day = 16;
     date.month = 10;
     date.year = 2011;
     System.out.println("DateOne: "+date.day+ "/" +date.month+
          "/" +date.year);
}
```

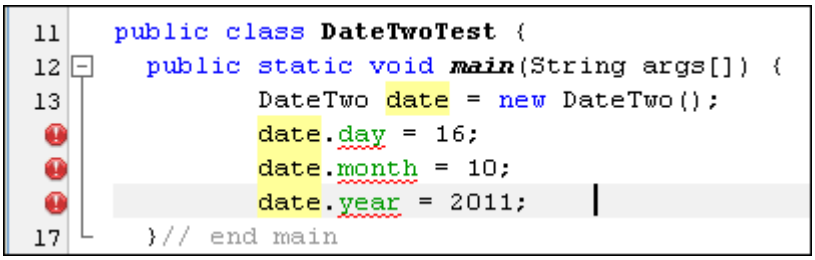
4. Save and compile your program.

5. Run the DateOneTest class to test the program.



6. Create another new Java Class called “DateTwo” similar to DateOne with three member fields (`day`, `month`, `year`).
7. Set the access modifier for the member fields to `private`.
8. Open the DateTwoTest class and perform the same steps as in Step 3, however in this case, create an instance of the DateTwo class instead of the DateOne class. The other lines of code remain the same.

   Note: NetBeans warns you with an error icon next to each line that references the fields of the DateTwo object.



```
11  public class DateTwoTest {
12    public static void main(String args[]) {
13          DateTwo date = new DateTwo();
            date.day = 16;
            date.month = 10;
            date.year = 2011;
17    }// end main
```

   Examine the warning message by putting your cursor over any of the red icons. It says that “`day` has private access in DateTwo” (similar message for each field). Although NetBeans lets you click Save without issuing a compiler error, it only saves the file. It does not compile the code or create the DateTwoTest.class file.

Accessing Encapsulated Attributes of a Class

In this task, you create a class with private attributes but enable them to be manipulated from another class.

9. Create a new Java Class called “DateThree” and add the same three private fields as the `DateTwo` class.
10. Add a `public get` method for the day field. This method should return an `int`. In the body of the method, return the day field. Example:

```
public int getDay(){
    return day;
}
```

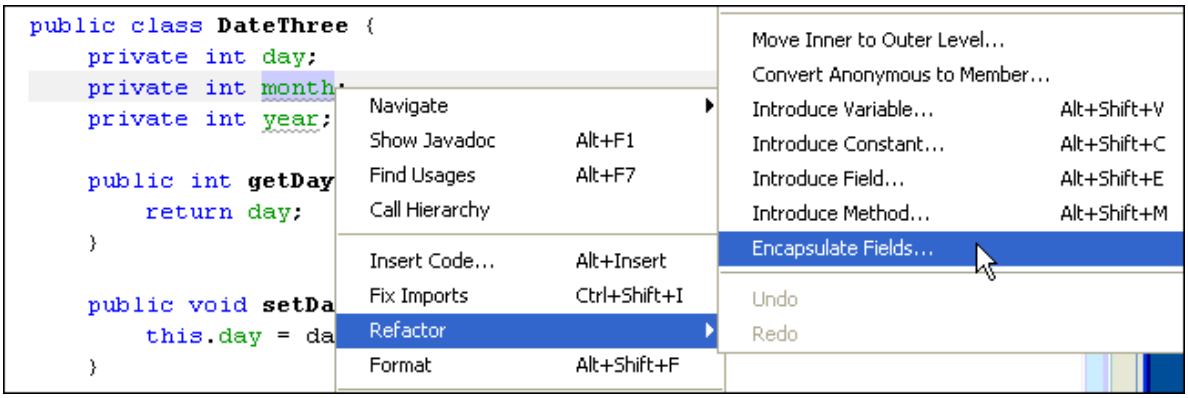
11. Add a `public set` method for the `day` field. This method returns `void` but takes an argument of type `int`. In the body of the method assign the argument to the `day` field. Example:

```
public void setDay(int day){
     this.day = day;
}
```

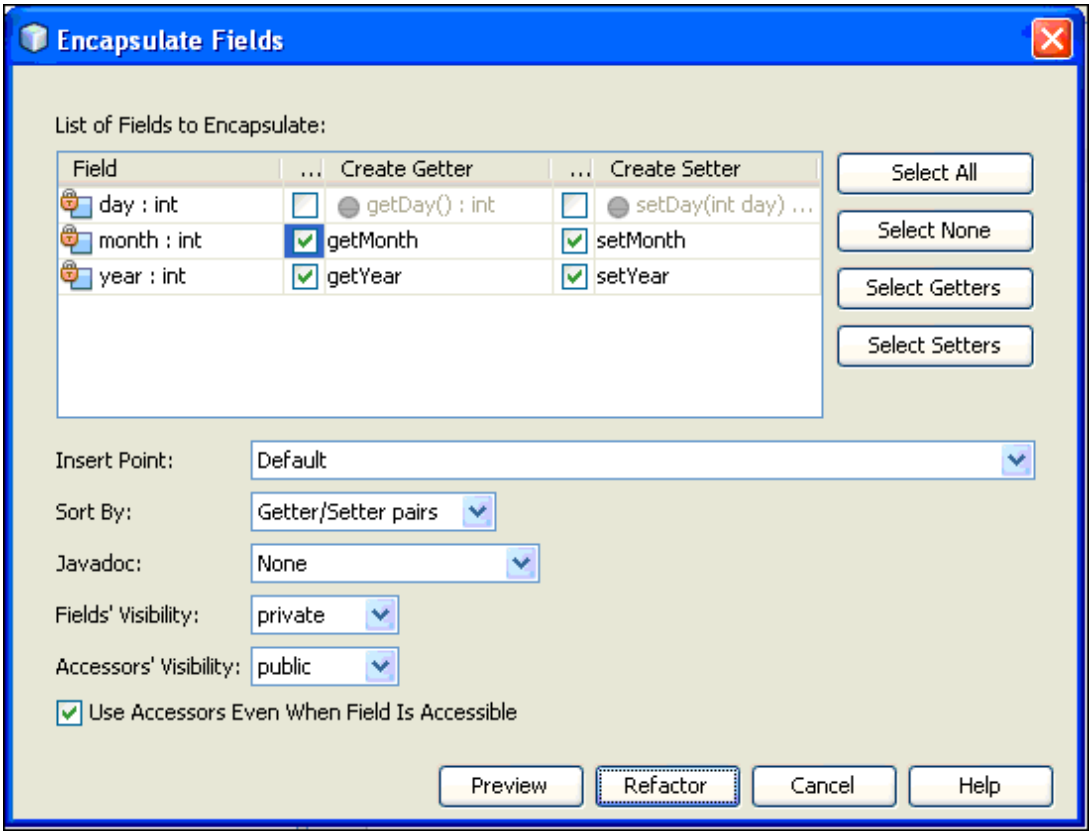
12. Add a similar `get` and `set` method for both the `month` and the `year` fields. Read the Hint below first to save some time.

Hint

Most IDEs automatically create the `get` and `set` methods for private fields in a class. This is part of a feature called “Refactoring”. In NetBeans, you can take advantage of this feature by selecting (highlighting) one of the private fields and right-clicking it. Select **Refactor > Encapsulate Fields** from the context menu.



The **Encapsulate Fields** window opens. Select the get and set method check boxes for the remaining fields. You may want to also set the **Javadoc** setting to **None** to streamline your method code. Click **Refactor** to close the window and create the methods.



```
public int getMonth() {
    return month;
}

public void setMonth(int month) {
    this.month = month;
}

public int getYear() {
    return year;
}

public void setYear(int year) {
    this.year = year;
}
```

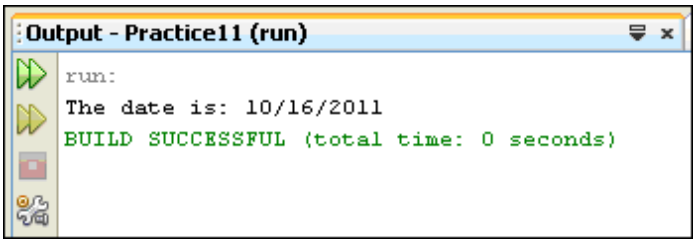
13. Open the DateThreeTest class.
14. In the `main` method, declare an object of type DateThree called “date”. Create an instance of the DateThree class.
15. Using the DateThree object reference, invoke the `setMonth`, `setDay`, and `setYear` methods of the DateThree object to set the three values of a date. Example:

```
date.setMonth(10);
date.setDay(16);
date.setYear(2011);
```

16. Complete the `main` method by displaying the entire date in the format of your choice. For example:

```
System.out.println(The date is: ”+ date.getMonth() +
  “/” + date.getDay() + “/” + date.getYear());
```

17. Save and compile your program. Run the DateThreeTest class to test it.



Challenge Practice 10-2: Adding Validation to the DateThree Class

This practice is optional. Check with your instructor for recommendations about which optional practices to do. Perform this practice only if you are certain that you have enough time to perform all of the non-optional practices.

Overview

In this practice, you add a `setDate` method to the DateThree class that performs validation on the date part values that are passed into the method.

Assumptions

This practice assumes that you have finished Practice 10-1.

Tasks

1. In the DateThree class, add a public `setDate` method that takes three arguments of type `int`. These values are assigned to the `day`, `month`, and `year` fields respectively, as in the following example:

   ```
   public void setDate(int d, int m, int y)
   ```

   Perform the validation indicated in the table below before assigning the argument values to the fields. Detailed steps are provided after the table.

Step	Code Description	Choices or Values
a.	Valid values for the `year` field	Between 1000 and 10000
b.	Valid values for the `month` field	1 – 12
c.	Valid values for the `day` field	30, 31, 28, 29 Depends on the month

   Note

   Use a `switch case` statement to determine the `month`. Use `if/else` statements to perform the validation. Display an error message if the value is invalid.

   a. In the `setDate` method, add the following `if/else` statement to check the validity of the year argument. Note: The `year` field is set to 0 in the case of an invalid year argument. You check for a 0 year value later.

```
if (y > 1000 && y < 10000){
    this.year = y;
} else {
    System.out.println(y + " is not a valid year.");
    this.year = 0;
}
```

   b. Create a `switch` statement that evaluates the month argument. Months 1, 3, 5, 7, 8, 10, and 12 have 31 days. Check for these values first in the `switch` statement. If the month argument equals any of these cases, assign the month argument to the `month` field, then include an `if/else` statement to test the value of the day argument. It should be between 1 and 31, inclusive. If it is not, display an error message and set the `day` field to `0`.

Example:

```
switch (m) {
     case 1:
     case 3:
     case 5:
     case 7:
     case 8:
     case 10:
     case 12:
           this.month = m;
           if (d > 0 && d < 32){
              this.day = d;
           } else {
             System.out.println(d + " is an invalid day for "
             + month);
             this.day = 0;
           }
           break;
//(switch statement continues in step c)
```

c. Use the following logic to complete the `switch` statement. In the `case block` for the month of February (case 2), you must also test for a leap year if the day argument is 29. The logic for the remaining months is similar to what you wrote for months containing 31 days.

```
...
case 2:
     this.month = m;
     if(d > 0 && d < 29) {
        this.day = d;
     } // check if year is a leap year when d==29 and m==2
     else if (d == 29){
        if(((y % 4 == 0) && !(y % 100 == 0)) || (y % 400 == 0)){
           this.day = d;
        } else {
           System.out.println("Invalid day. " +
            "Day cannot be 29 unless the year is a leap year.");
           this.day = 0;
        }  // end of inner if/else
     } // end of outer if/else
     break;
case 4:
case 6:
case 9:
case 11:
     this.month = m;
     if(d > 0 && d < 31){
        this.day = d;
     } else {
        System.out.println("Invalid day. Must be 1 to 30.");
        this.day = 0;
     }
     break;
default:
     System.out.println(m + " is an invalid month.");
     this.month = 0;
     break;
} // end switch
```

2. Add one more method called `displayDate`. In this method, first check for values of zero in `day`, `month`, or `year`. If any of these has a 0 value, print an "Invalid date" message. Otherwise, display the date using a date format of your choice. Example:

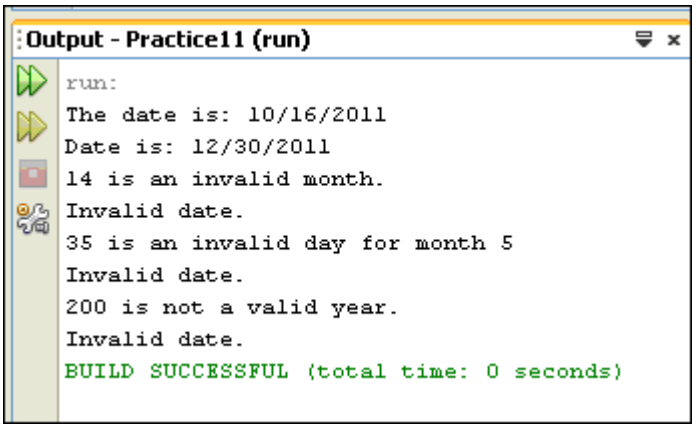
```
public void displayDate(){
     if(day == 0 || month == 0 || year == 0){
          System.out.println("Invalid date.");
     }
     else {
          System.out.println("Date is: " + month +"/"+
             day +"/"+ year);
     }
}
```

3. Open the DateThreeTest class and, using the `setDate` and `displayDate` methods, write code to perform the following tests:
   - Test with valid values for month, day and year
   - Test with invalid value for month 14
   - Test with invalid value for day 35
   - Test with invalid year 200

   Example for the first test:

```
date.setDate(30,12,2011);
date.displayDate();
```

4. Save and compile your program and run the DateThreeTest class. You should see an output similar to the following:



Compare the output to your code to match up the messages with the particular test that was run.

Practice 10-3: Creating Constructors to Initialize Objects

Overview

In this practice, you create a class and use constructors to initialize objects.

Assumptions

This practice assumes that the RectangleTest.java file appears in the practice folder for this lesson, `Lesson10`, and consequently also in the Practice11 project.

Tasks

1. Create a new Java Class called "Rectangle". Add two `private` fields of type `int` and name them `width` and `height`.
2. Add a constructor with no arguments (a "no args constructor"). The following table provides the high-level steps to create this constructor. If you need more help, use the detailed instructions below the table.

Step	Description	Choices or Values
a.	Syntax for declaring a no args constructor	`public <class_name>()`
b.	Initialize the private fields	`width = 25` `height = 10`
c.	Print a message	"Default rectangle created: width = 25, height = 10"

   a. In the Rectangle class, declare a `public` no args constructor as follows:

```
public Rectangle(){
}
```

   b. Assign the `width` field to the value 25 and the `height` field to the value 10.

   c. Use `System.out.println` to display the message shown in Step b of the table above.

3. Add a second constructor that accepts two `int` arguments: `w` and `h` (for "width" and "height"). The following table provides the high-level steps to complete this constructor. If you need more help, use the detailed instructions below the table.

Step	Code Description	Choices or Values
a.	Set height to `h` and width to `w` after validating the argument values	`h` and `w` should be > 0 and < 30
b.	Display a message for each condition	Error message if the numbers are invalid Message indicating that a rectangle has been created (show the height and width)

   a. In the constructor, add an `if/else` statement to ensure that the values passed into the constructor are within the acceptable range of 1 through 29. If both arguments are valid, assign the argument to its respective member field.

   b. After assigning the values, print a message that indicates that a rectangle has been created with the designated values. Include the `width` and `height` values in the message. If the argument values are not valid, display an error message.

Solution:

```
public Rectangle(int w, int h){
     if(w > 0 && w < 30 && h > 0 && h < 30){
          width = w;
          height = h;
          System.out.println("Rectangle created: width = "
            +width+ " and height = "+height);
     }
     else {
          System.out.println("Invalid width and/or height. "+
             "Each must be positive and less than 30.");
     }
}
```

4. Create a `getArea` method that calculates and returns the area of the rectangle (width * height).

Solution:

```
public int getArea(){
     return width * height;
}
```

5. Create a `draw` method that prints the rectangle shape, as determined by its width and height, in a series of rows containing asterisks (*). The following steps provide more detailed instructions:
   a. Create a nested `for` loop to draw the rectangle using asterisks.
   b. The outer `for` loop iterates through the rows of the rectangle. The number of rows corresponds to the value of the `height` field.
   c. The inner `for` loop iterates through the columns of each row. The number of columns corresponds to the value of the `width` field.

Solution:

```
public void draw(){
     for(int rowCounter=0;rowCounter<height;rowCounter++){
          for(int colCounter=0;colCounter<width;colCounter++){
             System.out.print("*");
          } // end of each row
          System.out.println();  // create a new line
     } // end of all rows
} // end of draw method
```

6. Save and compile your program.
7. Open the RectangleTest class. In the `main` method, declare and create two Rectangle objects, `r1` and `r2`, such that:
   - `r1` is created with the no args constructor
   - `r1` is drawn immediately after it is created (use the draw method)

- `r2` is created using the constructor with arguments
- `r2` is drawn and the area is printed

Solution:

```
public static void main(String args[]){
     // Rectangle with default values (no args)
     Rectangle r1 = new Rectangle();
     r1.draw();
     //Rectangle from args constructor
     Rectangle r2 = new Rectangle(15,5);
     System.out.println("Area of r2 is: "+r2.getArea());
     r2.draw();
}
```

8. Save and compile your program. Run the RectangleTest class to test it. The output should look similar to this: