1. `Employee` Class

Write a class named `Employee` that has the following fields:

- `name`. The `name` field references a `String` object that holds the employee's name.
- `idNumber`. The `idNumber` is an `int` variable that holds the employee's ID number.
- `department`. The `department` field references a `String` object that holds the name of the department where the employee works.
- `position`. The `position` field references a `String` object that holds the employee's job title.

The class should have the following constructors:

- A constructor that accepts the following values as arguments and assigns them to the appropriate fields: employee's name, employee's ID number, department, and position.
- A constructor that accepts the following values as arguments and assigns them to the appropriate fields: employee's name and ID number. The `department` and `position` fields should be assigned an empty string (`""`).
- A no-arg constructor that assigns empty strings (`""`) to the `name`, `department`, and `position` fields, and 0 to the `idNumber` field.

Write appropriate mutator methods that store values in these fields and accessor methods that return the values in these fields. Once you have written the class, write a separate program that creates three `Employee` objects to hold the following data:

Name	ID Number	Department	Position
Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

The program should store this data in the three objects and then display the data for each employee on the screen.

2. `Car` Class

Write a class named `Car` that has the following fields:

- **`yearModel`**. The `yearModel` field is an `int` that holds the car's year model.
- **`make`**. The `make` field references a `String` object that holds the make of the car.
- **`speed`**. The `speed` field is an `int` that holds the car's current speed.

In addition, the class should have the following constructor and other methods.

- Constructor. The constructor should accept the car's year model and make as arguments. These values should be assigned to the object's `yearModel` and `make` fields. The constructor should also assign 0 to the `speed` field.
- Accessors. Appropriate accessor methods should get the values stored in an object's `yearModel`, `make`, and `speed` fields.
- `accelerate`. The `accelerate` method should add 5 to the `speed` field each time it is called.
- `brake`. The `brake` method should subtract 5 from the `speed` field each time it is called.

Demonstrate the class in a program that creates a `Car` object, and then calls the `accelerate` method five times. After each call to the `accelerate` method, get the current speed of the car and display it. Then call the `brake` method five times. After each call to the `brake` method, get the current speed of the car and display it.

3. Personal Information Class

Design a class that holds the following personal data: name, address, age, and phone number. Write appropriate accessor and mutator methods. Demonstrate the class by writing a program that creates three instances of it. One instance should hold your information, and the other two should hold your friends' or family members' information.

4. `RetailItem` Class

Write a class named `RetailItem` that holds data about an item in a retail store. The class should have the following fields:

- `description`. The `description` field references a `String` object that holds a brief description of the item.
- `unitsOnHand`. The `unitsOnHand` field is an `int` variable that holds the number of units currently in inventory.
- `price`. The `price` field is a `double` that holds the item's retail price.

Write a constructor that accepts arguments for each field, appropriate mutator methods that store values in these fields, and accessor methods that return the values in these fields. Once you have written the class, write a separate program that creates three `RetailItem` objects and stores the following data in them:

	Description	Units on Hand	Price
Item #1	Jacket	12	59.95
Item #2	Designer Jeans	40	34.95
Item #3	Shirt	20	24.95

5. `Fee record` class

A class named `FeeRecord` contains fields for a student's name, enrolment number, annual fees, sports fees, and tuition fees. Write a constructor that accepts the two parameters name and enrolment number. Also write mutator and acceptor methods of the name `setFees` that accept the values of annual fees, sports fees, and tuition fees fields. Write a method `printStudentDetails` that displays the student's name and enrolment number. The class should have a method that returns the total fees by adding annual fees, sports fees, and tuition

fees. Write a program that demonstrates the class by creating a `FeeRecord` object, then asks the user to enter the data for a student. The program should also display the total fees amount.

6. `TestScores` Class

Design a `TestScores` class that has fields to hold three test scores. The class should have a constructor, accessor and mutator methods for the test score fields, and a method that returns the average of the test scores. Demonstrate the class by writing a separate program that creates an instance of the class. The program should ask the user to enter three test scores, which are stored in the `TestScores` object. Then the program should display the average of the scores, as reported by the `TestScores` object.

7. Total Sales

Design a class named `TotalSales` that holds the weekly sales records of two salespersons. The class must have a constructor, and accessor and mutator methods for storing the sales record information. Write a method that prints the weekly sales amounts and also which salesperson's performance is better by comparing each of their amounts. Demonstrate the class by writing a separate program that creates an instance of the `TotalSales` class. The program should ask the user to enter the weekly sales amount of two salespersons and store them in the `TotalSales` object. Then the program should display which salesperson has better sales records using the `TotalSales` object.

8. `Temperature` Class

Write a `Temperature` class that will hold a temperature in Fahrenheit, and provide methods to get the temperature in Fahrenheit, Celsius, and Kelvin. The class should have the following field:

- `ftemp` – A `double` that holds a Fahrenheit temperature.

The class should have the following methods:

- Constructor – The constructor accepts a Fahrenheit temperature (as a `double`) and stores it in the `ftemp` field.
- `setFahrenheit` – The `setFahrenheit` method accepts a Fahrenheit temperature (as a `double`) and stores it in the `ftemp` field.
- `getFahrenheit` – Returns the value of the `ftemp` field, as a Fahrenheit temperature (no conversion required).
- `getCelsius` – Returns the value of the `ftemp` field converted to Celsius.
- `getKelvin` – Returns the value of the `ftemp` field converted to Kelvin.

Use the following formula to convert the Fahrenheit temperature to Celsius:

$$Celsius = (5/9) \times (Fahrenheit - 32)$$

Use the following formula to convert the Fahrenheit temperature to Kelvin:

$$Kelvin = ((5/9) \times (Fahrenheit - 32)) + 273$$

Demonstrate the `Temperature` class by writing a separate program that asks the user for a Fahrenheit temperature. The program should create an instance of the `Temperature` class, with the value entered by the user passed to the constructor. The program should then call the object's methods to display the temperature in Celsius and Kelvin.

9. Days in a Month

Write a class named `MonthDays`. The class's constructor should accept two arguments:

- An integer for the month (1 = January, 2 February, etc.).
- An integer for the year

The class should have a method named `getNumberOfDays` that returns the number of days in the specified month. The method should use the following criteria to identify leap years:

1. Determine whether the year is divisible by 100. If it is, then it is a leap year if and if only it is divisible by 400. For example, 2000 is a leap year but 2100 is not.
2. If the year is not divisible by 100, then it is a leap year if and if only it is divisible by 4. For example, 2008 is a leap year but 2009 is not.

Demonstrate the class in a program that asks the user to enter the month (letting the user enter an integer in the range of 1 through 12) and the year. The program should then display the number of days in that month. Here is a sample run of the program:

```
Enter a month (1-12): 2 [Enter]
Enter a year: 2008 [Enter]
29 days
```

10. A Game of Twenty-One

For this assignment, you will write a program that lets the user play against the computer in a variation of the popular blackjack card game. In this variation of the game, two six-sided dice are used instead of cards. The dice are rolled, and the player tries to beat the computer's hidden total without going over 21.

Here are some suggestions for the game's design:

- Each round of the game is performed as an iteration of a loop that repeats as long as the player agrees to roll the dice, and the player's total does not exceed 21.
- At the beginning of each round, the program will ask the user whether or not he or she wants to roll the dice to accumulate points.
- During each round, the program simulates the rolling of two six-sided dice. It rolls the dice first for the computer, and then it asks the user whether he or she wants to roll. (Use the `Die` class that was shown in Code Listing 6-14 to simulate the dice.)
- The loop keeps a running total of both the computer's and the user's points.
- The computer's total should remain hidden until the loop has finished.
- After the loop has finished, the computer's total is revealed, and the player with the most points, without going over 21, wins.

11. Freezing and Boiling Points

The following table lists the freezing and boiling points of several substances.

Substance	Freezing Point	Boiling Point
Ethyl Alcohol	–173	172
Oxygen	–362	–306
Water	32	212

Design a class that stores a temperature in a `temperature` field and has the appropriate accessor and mutator methods for the field. In addition to appropriate constructors, the class should have the following methods:

- **`isEthylFreezing`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or below the freezing point of ethyl alcohol. Otherwise, the method should return `false`.
- **`isEthylBoiling`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or above the boiling point of ethyl alcohol. Otherwise, the method should return `false`.
- **`isOxygenFreezing`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or below the freezing point of oxygen. Otherwise, the method should return `false`.
- **`isOxygenBoiling`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or above the boiling point of oxygen. Otherwise, the method should return `false`.
- **`isWaterFreezing`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or below the freezing point of water. Otherwise, the method should return `false`.
- **`isWaterBoiling`.** This method should return the `boolean` value `true` if the temperature stored in the `temperature` field is at or above the boiling point of water. Otherwise, the method should return `false`.

Write a program that demonstrates the class. The program should ask the user to enter a temperature, and then display a list of the substances that will freeze at that temperature and those that will boil at that temperature. For example, if the temperature is –20 the class should report that water will freeze and oxygen will boil at that temperature.

12. `SavingsAccount` Class

Design a `SavingsAccount` class that stores a savings account's annual interest rate and balance. The class constructor should accept the amount of the savings account's starting balance. The class should also have methods for subtracting the amount of a withdrawal, adding the amount of a deposit, and adding the amount of monthly interest to the balance. The monthly interest rate is the annual interest rate divided by twelve. To add the monthly interest to the balance, multiply the monthly interest rate by the balance, and add the result to the balance.

Test the class in a program that calculates the balance of a savings account at the end of a period of time. It should ask the user for the annual interest rate, the starting balance, and

the number of months that have passed since the account was established. A loop should then iterate once for every month, performing the following:

a. Ask the user for the amount deposited into the account during the month. Use the class method to add this amount to the account balance.
b. Ask the user for the amount withdrawn from the account during the month. Use the class method to subtract this amount from the account balance.
c. Use the class method to calculate the monthly interest.

After the last iteration, the program should display the ending balance, the total amount of deposits, the total amount of withdrawals, and the total interest earned.

13. Deposit and Withdrawal Files

Use Notepad or another text editor to create a text file named `Deposits.txt`. The file should contain the following numbers, one per line:

```
100.00
124.00
78.92
37.55
```

Next, create a text file named `Withdrawals.txt`. The file should contain the following numbers, one per line:

```
29.88
110.00
27.52
50.00
12.90
```

The numbers in the `Deposits.txt` file are the amounts of deposits that were made to a savings account during the month, and the numbers in the `Withdrawals.txt` file are the amounts of withdrawals that were made during the month. Write a program that creates an instance of the `SavingsAccount` class that you wrote in Programming Challenge 12. The starting balance for the object is 500.00. The program should read the values from the `Deposits.txt` file and use the object's method to add them to the account balance. The program should read the values from the `Withdrawals.txt` file and use the object's method to subtract them from the account balance. The program should call the class method to calculate the monthly interest, and then display the ending balance and the total interest earned.

14. Dice Game

Write a program that uses the `Die` class that was presented in this chapter to play a simple dice game between the computer and the user. The program should create two instances of the `Die` class (each a 6-sided die). One `Die` object is the computer's die, and the other `Die` object is the user's die.

The program should have a loop that iterates 10 times. Each time the loop iterates, it should roll both dice. The die with the highest value wins. (In case of a tie, there is no winner for that particular roll of the dice.)

As the loop iterates, the program should keep count of the number of times the computer wins, and the number of times that the user wins. After the loop performs all of its iterations, the program should display who was the grand winner, the computer or the user.

15. Roulette Wheel Colors

On a roulette wheel, the pockets are numbered from 0 to 36. The colors of the pockets are as follows:

- Pocket 0 is green.
- For pockets 1 through 10, the odd numbered pockets are red and the even numbered pockets are black.
- For pockets 11 through 18, the odd numbered pockets are black and the even numbered pockets are red.
- For pockets 19 through 28, the odd numbered pockets are red and the even numbered pockets are black.
- For pockets 29 through 36, the odd numbered pockets are black and the even numbered pockets are red.

Write a class named `RoulettePocket`. The class's constructor should accept a pocket number. The class should have a method named `getPocketColor` that returns the pocket's color, as a string.

Demonstrate the class in a program that asks the user to enter a pocket number, and displays whether the pocket is green, red, or black. The program should display an error message if the user enters a number that is outside the range of 0 through 36.

16. Coin Toss Simulator

Write a class named `Coin`. The `Coin` class should have the following field:

- A `String` named `sideUp`. The `sideUp` field will hold either "heads" or "tails" indicating the side of the coin that is facing up.

The `Coin` class should have the following methods:

- A no-arg constructor that randomly determines the side of the coin that is facing up ("heads" or "tails") and initializes the `sideUp` field accordingly.
- A `void` method named `toss` that simulates the tossing of the coin. When the toss method is called, it randomly determines the side of the coin that is facing up ("heads" or "tails") and sets the `sideUp` field accordingly.
- A method named `getSideUp` that returns the value of the `sideUp` field.

Write a program that demonstrates the `Coin` class. The program should create an instance of the class and display the side that is initially facing up. Then, use a loop to toss the coin 20 times. Each time the coin is tossed, display the side that is facing up. The program should keep count of the number of times heads is facing up and the number of times tails is facing up, and display those values after the loop finishes.

17. Tossing Coins for a Dollar

For this assignment you will create a game program using the `Coin` class from Programming Challenge 16. The program should have three instances of the `Coin` class: one representing a quarter, one representing a dime, and one representing a nickel.

When the game begins, your starting balance is $0. During each round of the game, the program will toss the simulated coins. When a coin is tossed, the value of the coin is added to your balance if it lands heads-up. For example, if the quarter lands heads-up, 25 cents is

added to your balance. Nothing is added to your balance for coins that land tails-up. The game is over when your balance reaches one dollar or more. If your balance is exactly one dollar, you win the game. You lose if your balance exceeds one dollar.

18. Fishing Game Simulation

For this assignment, you will write a program that simulates a fishing game. In this game, a six-sided die is rolled to determine what the user has caught. Each possible item is worth a certain number of fishing points. The points will remain hidden until the user is finished fishing, and then a message is displayed congratulating the user, depending on the number of fishing points gained.

Here are some suggestions for the game's design:

- Each round of the game is performed as an iteration of a loop that repeats as long as the player wants to fish for more items.
- At the beginning of each round, the program will ask the user whether or not he or she wants to continue fishing.
- The program simulates the rolling of a six-sided die (use the `Die` class that was shown in Code Listing 6-14).
- Each item that can be caught is represented by a number generated from the die; for example, 1 for "a huge fish", 2 for "an old shoe", 3 for "a little fish", and so on.
- Each item the user catches is worth a different amount of points.
- The loop keeps a running total of the user's fishing points.
- After the loop has finished, the total number of fishing points is displayed, along with a message that varies depending on the number of points earned.