Loop

Han-fen Hu

Outline

- Increment and Decrement Operators
- **`while`** Loop
- **`do-while`** Loop
- **`for`** Loop
- Nested Loops
- Deciding Which Loop to Use

Increment and Decrement Operators (1)

- There are numerous times where a variable must simply be incremented or decremented.

```
number = number + 1;
number = number – 1;
```

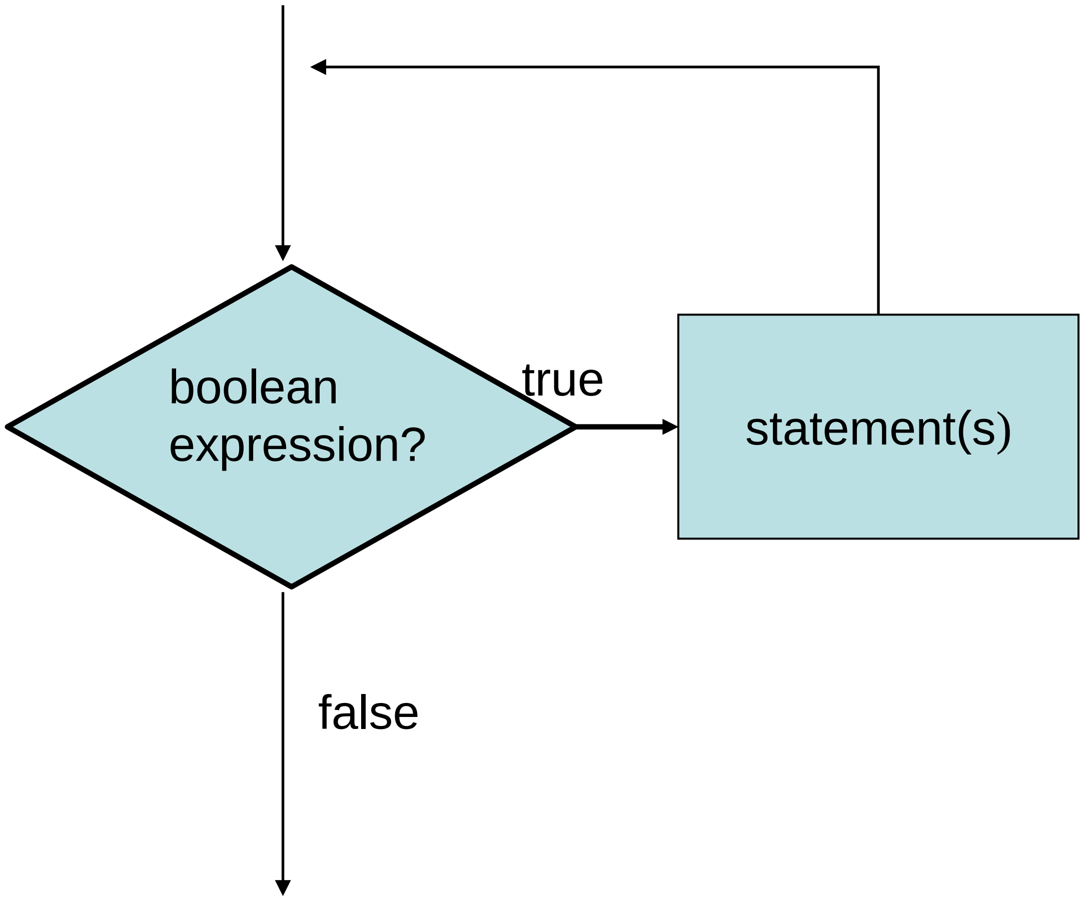
- Java provide shortened ways to increment and decrement a variable’s value.
- Using the **++** or **--** unary operators, this task can be completed quickly.

```
number++;  or  ++number;
number--;  or  --number;
```

Increment and Decrement Operators (2)

- When an increment or decrement are the only operations in a statement, there is no difference between prefix and postfix notation.
- When used in an expression:
  - prefix notation indicates that the variable will be incremented or decremented prior to the rest of the equation being evaluated.
  - postfix notation indicates that the variable will be incremented or decremented after the rest of the equation has been evaluated.

`while` loop Flowchart



do-while Loop Flowchart

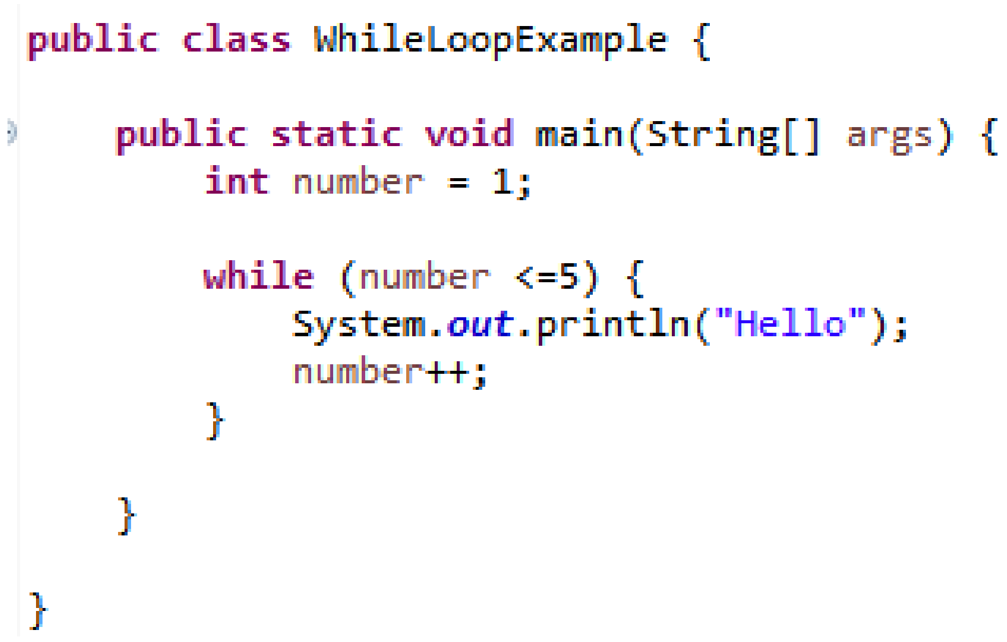
`while` Loop (1)

```
while (condition){
    statement1;
    statement2;
    statement3;
}
```

- While the condition is true, the statements will execute repeatedly.
  - The **`while`** loop is a *pretest* loop, which means that it will test the value of the condition prior to executing the loop.
  - A **`while`** loop executes 0 or more times. If the condition is false, the loop will not execute.
- Curly braces are required to enclose block statement while loops.

Question

How many times of “Hello” would the following program print?



```
public class WhileLoopExample {

    public static void main(String[] args) {
        int number = 1;

        while (number <=5) {
            System.out.println("Hello");
            number++;
        }

    }

}
```

`while` Loop (2)

- Loops that do not end are called *infinite loops.*
  - Care must be taken to set the condition to false somewhere in the loop so the loop will end.

```
int x = 20;
while(x > 0){
   System.out.println("x is greater than 0");
}
```

- The variable **`x`** never gets decremented so it will always be greater than 0.
- How to fix the problem?

`while` Loop (3)

- This version of the loop decrements **`x`** during each iteration:

```
int x = 20;
while(x > 0){
   System.out.println("x is greater than 0");
   x--;
}
```

`while` Loop (4): Example

- *Input validation*

```
System.out.print("Enter a number in the " +
                 "range of 1 through 100: ");
number = keyboard.nextInt();
// Validate the input.
while (number < 1 || number > 100){
  System.out.println("That number is invalid.");
  System.out.print("Enter a number in the " +
                   "range of 1 through 100: ");
  number = keyboard.nextInt();
}
```

Lab (1)

- SoccerTeams.java
  - Ask the user to enter the number of players per team (between 9 and 15), and the available number of players. The program will return how many teams the players and form and how many players are left.

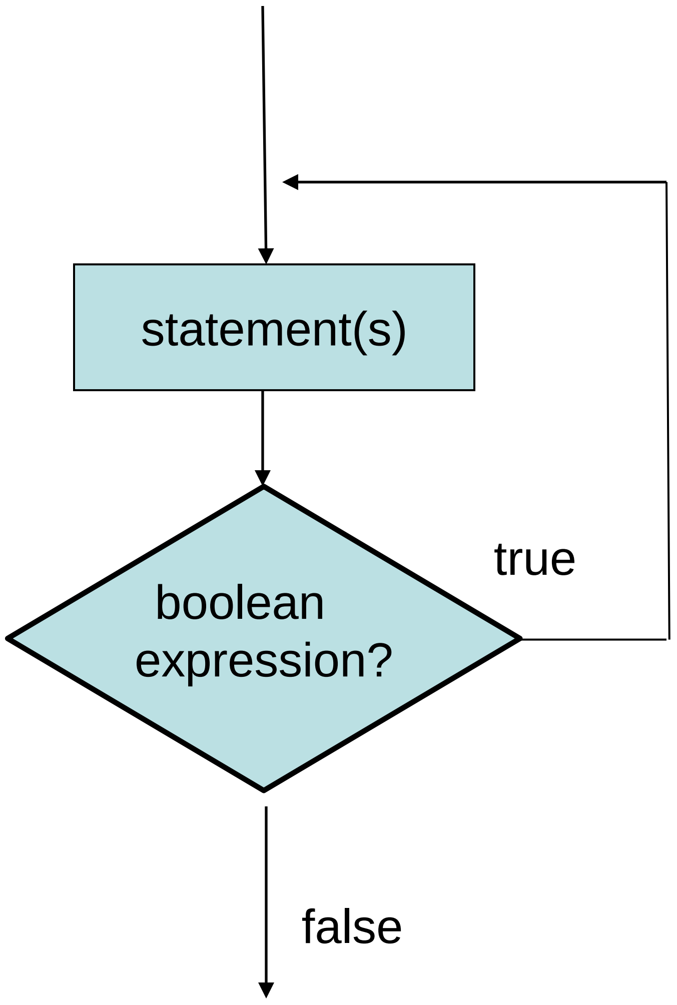
`do-while` Loop

```
do {
  statement(s);
}while (condition);
```

- While the condition is true, the statements will execute repeatedly.
  - The **`do-while`** loop is a *post-test* loop, which means it will execute the loop prior to testing the condition.
  - A **`do-while`** loop executes 1 or more times. If the condition is false, the loop will not execute.

UNLV

`do-while` Loop Flowchart



While Loop Flowchart

Exercise (1)

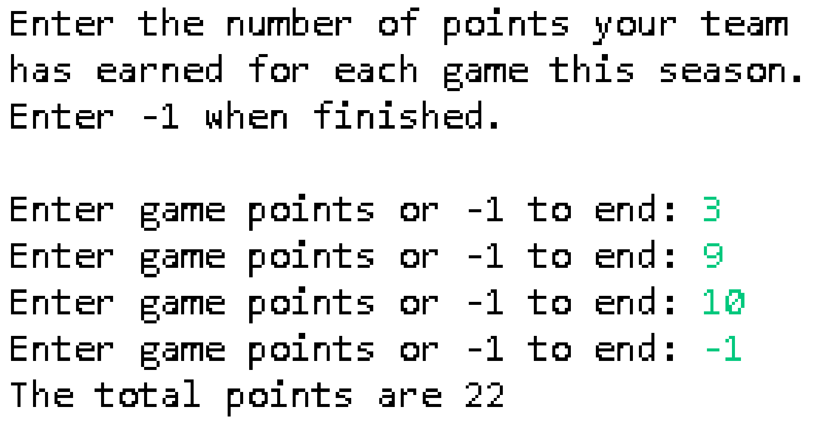
- SoccerTeams.java
  - Can we use a do-while Loop instead of while Loop in this program? If so, how? If not, why not?

Sentinel Values

- Sometimes the end point of input data is not known.
- A sentinel value can be used to notify the program to stop acquiring input.
- If it is a user input, the user could be prompted to input data that is not normally in the input data range (i.e. –1 where normal input would be positive.)
- Programs that get file input typically use the end-of-file marker to stop acquiring input data.

Lab (2)

- SoccerPoints.java
  - Calculate the total number of points a soccer team has earned over a series of games. The user enters a series of point values, then -1 when finished.



```
Enter the number of points your team
has earned for each game this season.
Enter -1 when finished.

Enter game points or -1 to end: 3
Enter game points or -1 to end: 9
Enter game points or -1 to end: 10
Enter game points or -1 to end: -1
The total points are 22
```

`for` Loop

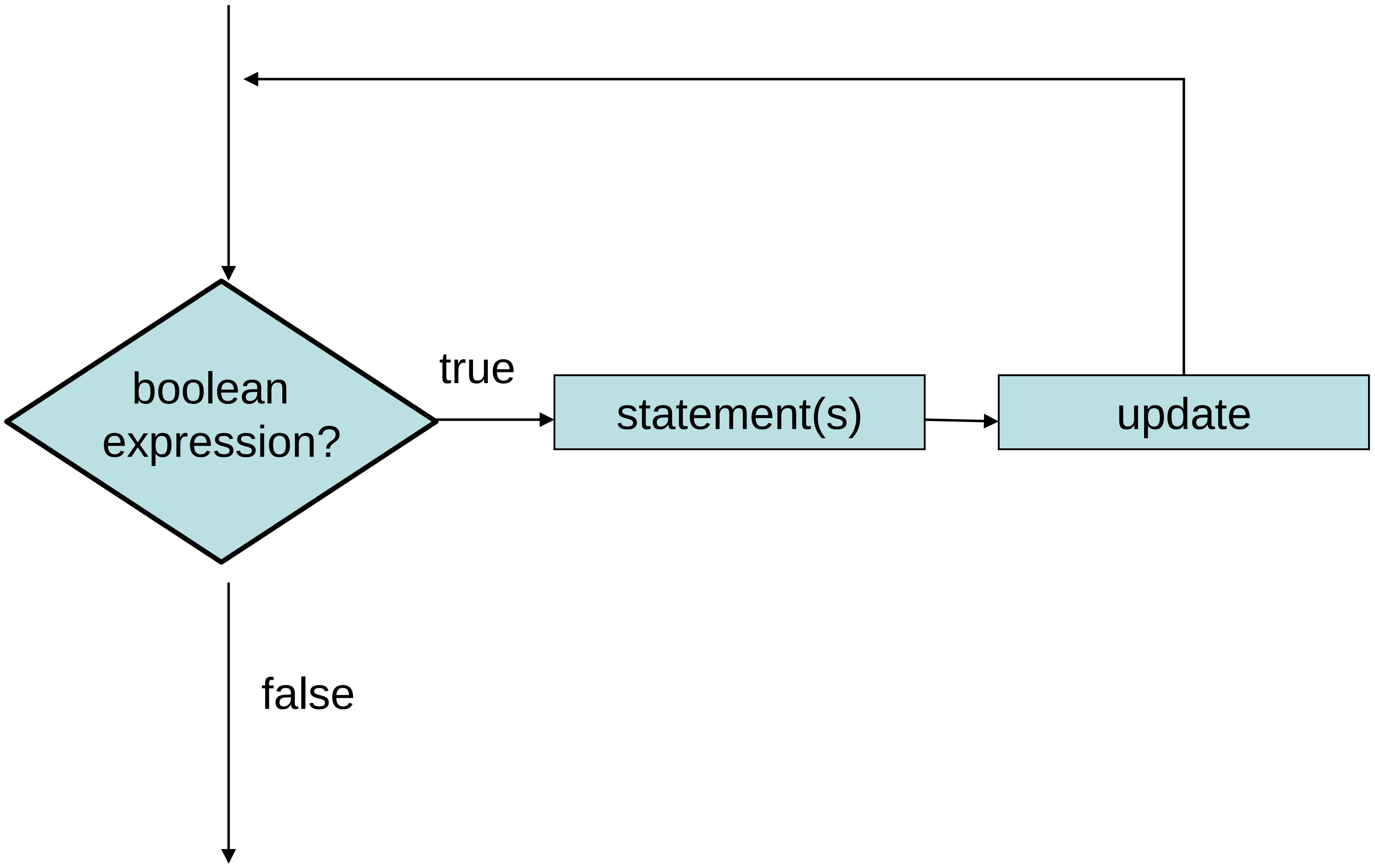
```
for(initialization; test; update){
   statement(s);
}
```

- The **`for`** loop allows the programmer to initialize a control variable, test a condition, and modify the control variable all in one line of code.
- The **`for`** loop is a *pre-test* loop.
- Typically, **`for`** loops initialize a counter variable that will be tested by the test section of the loop and updated by the update section.

`for` Loop (2)

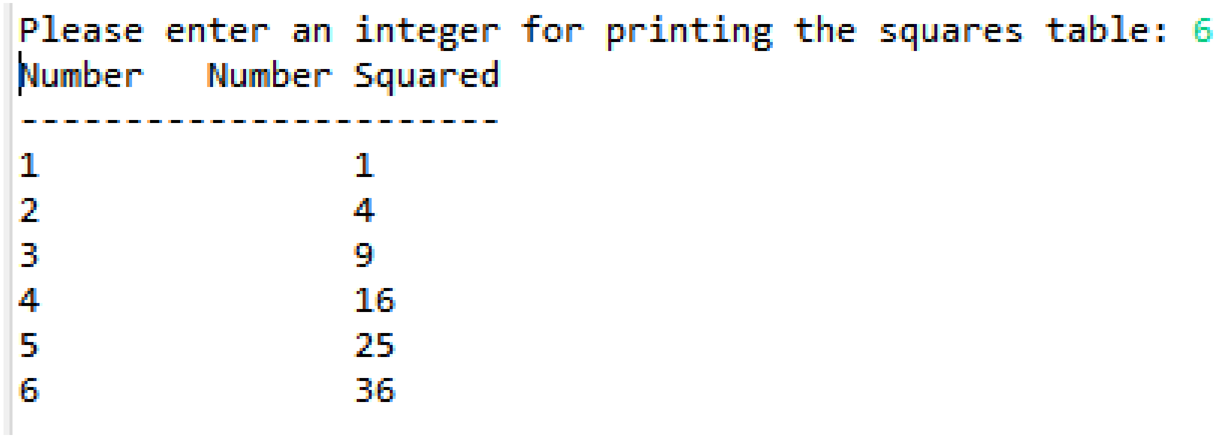
- The *initialization section* of the **`for`** loop allows the loop to initialize its own control variable.
  - The initialization section can initialize multiple variables.
  - Variables declared in this section have scope only for the **`for`** loop.
- The *test section* of the **`for`** statement acts in the same manner as the condition section of a **`while`** loop.
  - If left out, the test section defaults to true.
- The *update section* of the **`for`** loop is the last thing to execute at the end of each loop.
  - The update expression is usually used to increment or decrement the counter variable(s) declared in the initialization section of the for loop.
  - The update section may update multiple variables.

`for` Loop Flowchart



Lab (3)

- Squares.java
  - This program asks the user to enter an integer and then prints out the numbers and their squared numbers.



```
Please enter an integer for printing the squares table: 6
Number    Number Squared
-------------------------
1                1
2                4
3                9
4                16
5                25
6                36
```

`for` Loop (3)

- Te **`for`** loop may initialize and update multiple variables.

```
for(int i = 5, int j = 0;
    i < 10 || j < 20;
    i++, j+=2){
        statement(s);
}
```

Question: When Does it stop?

```
for(int i = 5, int j = 0;
   i < 10 || j < 20;
   i++, j+=2){
```

Iteration	value of i	value of j	i<10	j<20	i < 10 || j < 20
1	5	0	true	true	true
2	6	2	true	true	true
3	7	4	true	true	true
4	8	6	true	true	true
5	9	8	true	true	true
6	10	10	false	true	true
7	11	12	false	true	true
8	12	14	false	true	true
9	13	16	false	true	true
10	14	18	false	true	true
11	15	20	false	false	false

UNLV

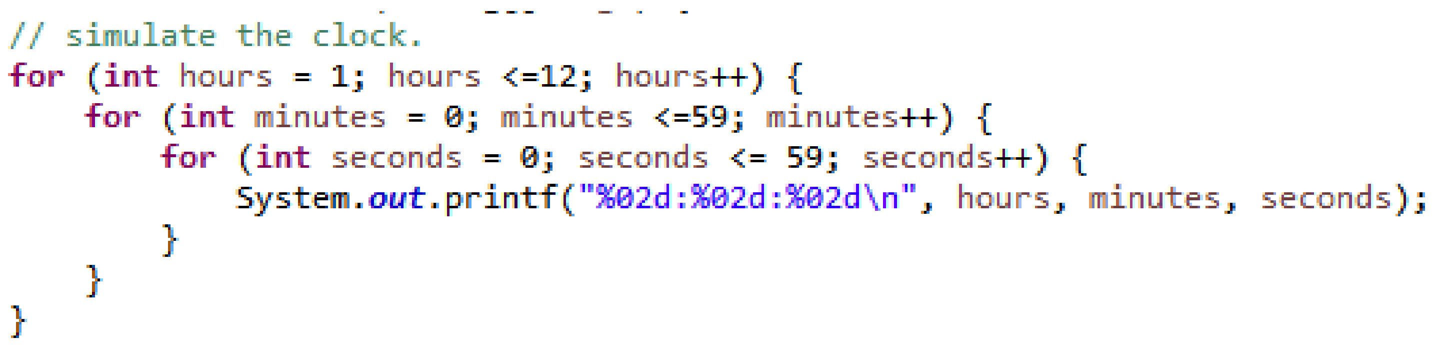
Nested Loops

- Loops can be nested.
- If a loop is nested, the inner loop will execute all of its iterations for each time the outer loop executes once.

```
for(int i = 0; i < 10; i++)
    for(int j = 0; j < 10; j++)
        loop statements;
```

  – The loop statements in this example will execute 100 times.

Nested Loops: Example



```java
// simulate the clock.
for (int hours = 1; hours <=12; hours++) {
    for (int minutes = 0; minutes <=59; minutes++) {
        for (int seconds = 0; seconds <= 59; seconds++) {
            System.out.printf("%02d:%02d:%02d\n", hours, minutes, seconds);
        }
    }
}
```

d : decimal integer [byte, short, int, long]
f : floating-point number [float, double]
c : character Capital C will uppercase the letter
s : String Capital S will uppercase all the letters in the string

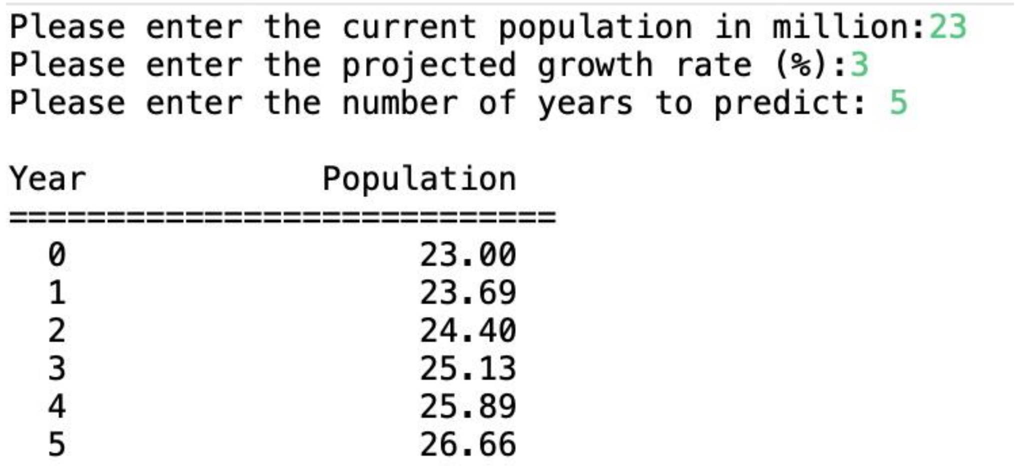
Deciding Which Loops to Use

- The **`while`** loop:
  - Pretest loop
  - Use it where you do not want the statements to execute if the condition is false in the beginning.
- The **`do-while`** loop:
  - Post-test loop
  - Use it where you want the statements to execute at least one time.
- The **`for`** loop:
  - Pretest loop
  - Use it where there is some type of counting variable that can be evaluated.

UNLV

Exercise (2)

- City Population
  - Write a program to calculate the projected city population. The user will enter the current population, projected growth rate, and the number of years to predict. The program prints the projected population for each year.
  - Note: You can use printf() for formatted result. For example, System.out.printf("%3d %10.2f", year, population);
    - %3d for three digits of aninteger
    - %10.2f for a float number with 2 digits after decimal point and a total length of 10 digits



```
Please enter the current population in million:23
Please enter the projected growth rate (%):3
Please enter the number of years to predict: 5

Year             Population
============================
  0                   23.00
  1                   23.69
  2                   24.40
  3                   25.13
  4                   25.89
  5                   26.66
```

File Handling

Outline

- Write Data to a File
- Appending Text to a File
- Read Data from a File
- Detecting the End of a File

File Input and Output

- Files can be input files or output files.
  - Data can be saved to a file.
  - Data can be read from a file.
- Steps
  - Files have to be opened.
  - Data is then written to the file, or read from the file.
  - The file must be closed prior to program termination.

`PrintWriter` Class (1)

- To open a file for text output you create an instance of the **`PrintWriter`** class.

Object Name

Constructor

```
PrintWriter outputFile = new PrintWriter("StudentData.txt");
```

Pass the name of the file that you wish to open as an argument to the PrintWriter constructor.

Warning: if the file already exists, it will be erased and replaced with a new file.

`PrintWriter` Class (2)

- The `PrintWriter` class allows you to write data to a file using the `print` and `println` methods, as you have been using to display data on the screen.
  - The `println` method of the `PrintWriter` class will place a newline character after the written data.
  - The `print` method writes data without writing the newline character.

`PrintWriter` Class (3)

Open the file.

```
PrintWriter outputFile = new PrintWriter("Names.txt");
outputFile.println("Chris");
outputFile.println("Kathryn");
outputFile.println("Jean");
outputFile.close();
```

Close the file.

Write data to the file.

PrintWriter Class (4)

- To use the **PrintWriter** class, put the following **import** statement at the top of the source file:

```
import java.io.*;
```

Specifying a File Location

- On a Windows computer, paths contain backslash (\) characters.
- If the backslash is used in a string literal, it is the escape character so you must use two of them:

```
PrintWriter outFile =
     new PrintWriter("D:\\PriceList.txt");
```

UNLV

Exceptions (1)

- When something unexpected happens in a Java program, an *exception* is thrown.
  - The method that is executing when the exception is thrown must either handle the exception or pass it up the line.
  - Handling the exception will be discussed later.
  - To pass it up the line, the method needs a **`throws`** clause in the method header.

Exceptions (2)

- To insert a **`throws`** clause in a method header, simply add the word *throws* and the name of the expected exception.
- **`PrintWriter`** objects can throw an **`IOException`**, so we write the **`throws`** clause like this:

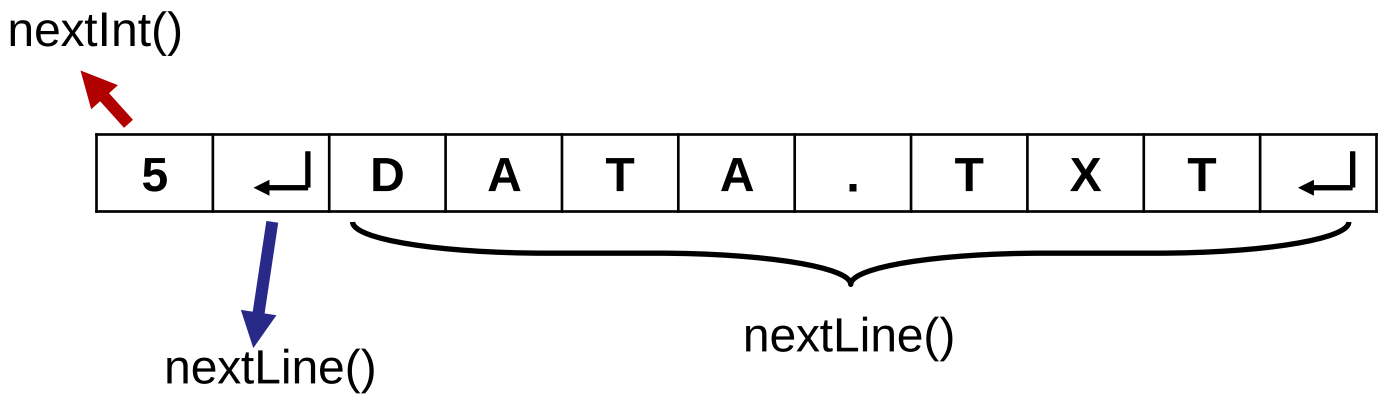
```
public static void main(String[] args) throws
   IOException
```

Lab (4)

- FileWriteDemo.java
  - Save a list of names to a file. The file name is designated by the user.
    The program prompts the user to (1) enter the number of names, (2) give a file name, and then (3) use a for loop to get the names from the user, and write the names to a file.

Consuming the Additional Character in Scanner object

- After using nextInt() to get an integer value, if we need to get the next value as a String, please make sure to consume the newline character before moving on



Appending Text to a File

- To avoid erasing a file that already exists, create a **`FileWriter`** object in this manner:

```
FileWriter fw =
     new FileWriter(filename, true);
```

OR

```
FileWriter fw =
     new FileWriter("names.txt", true);
```

- Then, create a **`PrintWriter`** object in this manner:

```
PrintWriter outputFile = new PrintWriter(fw);
```

Exercise (3)

- WriteFileDemo.java
  - Please revise the WriteFileDemo.java. When the file already exists, the new names entered will be appended to the file.

Reading Data From a File (1)

- You use the **`File`** class and the **`Scanner`** class to read data from a file:

Pass the name of the file as an argument to the `File` class constructor.

```
File myFile = new File("Customers.txt");
Scanner inputFile = new Scanner(myFile);
```

Pass the `File` object as an argument to the `Scanner` class constructor.

Reading Data From a File (2)

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the filename: ");
String filename = keyboard.nextLine();
File file = new File(filename);
Scanner inputFile = new Scanner(file);
```

- The lines above:
  - Creates an instance of the **`Scanner`** class to read from the keyboard
  - Prompt the user for a filename
  - Get the filename from the user
  - Create an instance of the **`File`** class to represent the file
  - Create an instance of the **`Scanner`** class that reads from the file

Reading Data From a File (3)

- Once an instance of **`Scanner`** is created, data can be read using the same methods that you have used to read keyboard input (**`nextLine`**, **`nextInt`**, **`nextDouble`**, etc).

```
// Open the file.
File file = new File("Names.txt");
Scanner inputFile = new Scanner(file);
// Read a line from the file.
String str = inputFile.nextLine();
// Close the file.
inputFile.close();
```

Exceptions

- The **`Scanner`** class can throw an **`IOException`** when a **`File`** object is passed to its constructor.
- So, we put a **`throws IOException`** clause in the header of the method that instantiates the **`Scanner`** class

Lab (5)

- ReadFileDemo.java

Detecting the End of a File

- The **`Scanner`** class's **`hasNext()`** method will return true if another item can be read from the file.

```
// Open the file.
File file = new File(filename);
Scanner inputFile = new Scanner(file);
// Read until the end of the file.
while (inputFile.hasNext()){
   String str = inputFile.nextLine();
   System.out.println(str);
}
inputFile.close();// close the file when done.
```

Exercise (4)

- ReadFileDemo.java
  - Please revise the ReadFileDemo.java, so that it can print all the content of the file.