Java

by Chandrasekaran Janardhanan

Contents

Install Notepad ++

- Open https://notepad-plus-plus.org/downloads/ in browser
- Click on the latest release
- Click on "Installer" link under "Download 64-bit x64" section to download the installer
- Execute the installer, follow the instructions to complete the installation.
- This text editor will be used to learn Java in the initial stages. Later an IDE will be introduced.
- Follow steps below to setup font in Notepad++
    - Open Notepad++
    - In Menu, go to Settings > Style Configurator
    - Select font name as "Consolas" and font size as 12
    - Check the box next to "Enable global font"
    - Check the box next to "Enable global font size"

Download JDK

- Go to https://jdk.java.net/
- Click the JDK under the "Ready for use" section
- Click on the link "zip" adjacent to windows/x64

Install JDK

- Open the downloaded zip file
- Right click on the folder starting with "jdk" and select "Copy"
- Paste the folder to D:
- Open Windows File Explore and navigate to the folder D:\jdk-[ver]\bin
- Click on the address bar and copy the folder location

Setup JDK

- Click "Start" button in windows
- Type "environment" and select "Edit the system environment variables"
- Click "Environment Variables"
- In "System Variables" section scroll down and find the variable named "Path"
- Select and click "Edit"
- Click "New"
- Paste the path copied after installation
- Move the path to the top using "Move Up" button
- Click OK three times on the respective window
- Click "Start" button and type "cmd"
- Select the Command Prompt option and open the command line window
- Type command "javac -version" and press enter
- If version number is displayed then JDK is successfully installed

Hello World in Java

Problem

Write a Java program to display "Hello World" in console.

Solution

- Create a folder "java-learn" in "D:". We will use this folder to hold all the Java programs that we will write while learning the programming language.
- Open Notepad++
- Click View in Menu and select "Folder as Workspace"
- This will display a window in left hand side
- Right click within this window and select "Add"
- Select the "D:\java-learn" folder. This helps in easier navigation of files.
- Select File > New to open a new window
- Type the following code:

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!!!");
    }
}
```

- Save the file with name HelloWorld.java in "D:\java-learn" folder.
- Go to "D:\java-learn" folder in Windows Explorer and type "cmd" in the address bar. This will open the command prompt in the "D:\java-learn" folder.
- Execute the following command to compile the program:

  javac HelloWorld.java
- If there are compilation errors reported rectify the errors.
- If there is no error displayed and the command prompt is shown, check if a file named "HelloWorld.class" is generated. This is called the byte code of a Java program.
- Execute the following command to execute the program:

  java HelloWorld
- Check if "HelloWorld!!" is displayed in the console, which means that the program has executed successfully.

Activity – Java source file name should match with class name

- Open "HelloWorld.java" in Notepad++
- In the first line of the program, change the class name "HelloWorld" to "World"
- Save the file
- Compile the program using "javac HelloWorld.java" command
- This will result in compilation error
- The compilation error message recommends us to change the source file name similar to the class name.

Activity – Understanding the importance of main() method

- Open "HelloWorld.java" in Notepad++
- The second line of the program defines a method within the HelloWorld class
- The name of this method is main() with parameter of String array named args[]
- Change the program name from "main" to "mainMethod"
- Compile the program using "javac HelloWorld.java" command
- Execute the program using "java HelloWorld"
- The execution of the program fails with error message mentioning that it cannot find main() method.
- Change back the method name as "main", compile and run the program.
- The key takeaway is that, the "java" command looks for method named main() and executes the lines within the main() method.

Activity – Java is case sensitive

- Open "HelloWorld.java" in Notepad++
- Change the main method parameter type to "string"
- Compile the program
- The compilation will fail with error message that it cannot find the symbol "string".
- Change the back the parameter to "String"

Activity – Understanding String and System class

- Open the link below in the browser:
  https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
- The data type String represented in the method parameter is a built-in class within Java library
- System referred in line 3 is another built-in class in Java, open the link below:
  https://docs.oracle.com/javase/8/docs/api/java/lang/System.html
- Classes String and System are built-in classes of Java, inline with that HelloWorld is a class that we have created.
- In the System documentation, click on the PrintStream link in "Field Summary" section
- Press Ctrl+F and search for the keyword "println(String x)". This is a method that is invoked in the line 3 to print the string inside the console.
- "println()" is a method defined in a built-in class, main() is a method that we have defined in our HelloWorld class.

Explanation for the program

- HelloWorld is the class name
- The class name should match with the file name.
- Java source files are stored with extension .java.
- class, public, static, void are Java keywords
- "class" keyword helps in defining a class

- main() is a method definition within HelloWorld
- The "java" command looks for existence of a main() method in the class and executes the main method.
- String args[] is a parameter for main method. "args" is the parameter name, "String" is a built-in class in Java to represent text.
- "System" is a built-in class. This class represents the computer system in which the Java program is running.
- "out" reference to the standard output device, which is the monitor.
- println() is a method in "out", which displays content on the console.
- Each executable line is separated by semi-colon
- Java is case sensitive
- String values are represented within double quotes
- Each section of program is enclosed within an open and closing curly braces. This is also called as block.
- There should be an indent in the next line of open curly braces
- Reduce the indent on closing curly braces
- "javac" is the command to compile a Java program
- "javac" command creates the class file of a java source file
- "java" is the command to execute a Java program

Questions

- What is the command to compile a Java program?
- What is the output generated by "javac" command?
- What is the command to execute a Java program?
- Java is not case sensitive. [True / False]
- What is the importance of class keyword in Java?
- What method does "java" command expects in a class to execute?
- What is String class?
- What is System class?
- What is the purpose of println() method?

Java and Platform Independence

Activity

Follow below steps to understand how Java is Platform Independent. [Links open reference images]

- Back Panel of a desktop PC – Click the link and open the diagram that represents the various ports available in the back panel of a desktop PC
- Various types of Ports in Back Panel – this diagram represents specific details about the ports:
  - Keyboard & Mouse Ports
  - Serial Port is for the monitor
  - LAN Ports
  - USB Ports
  - Speakers and Microphone
- Motherboard – this diagram represents a motherboard. Notice the ports available on top right corner. "CPU Socket" is where the processor is placed.
- Intel 8080 Processor – this represents the actual CPU with pin slots.
- Intel 8080 Processor Pins
  - D_0 to D_7 represents data bus
  - A_0 to A_{15} represents address bus
  - Each pin the processor sends either 1 or 0
- Components of Intel 8080 Processor – Refer the diagram in "Registers" section, represents the various registers within the processor.
- Example Assembly Instructions for Intel 8080 Processor
  - "memcpy" is a function
  - "memcpy" and "loop:" are called labels
  - "mov", "ora", "rz", "ldax", "mov", "inx", "dcx", "jnz", "ret" are instructions of the processor
  - "a", "b", "c", "d", "m", "h" represents the registers of the processor
- Refer sample assembly code to printf value in a console
- Based on the processor the pins and instruction set varies.
- As part of C compilation process, the source file is converted into Assembly Code specific to a processor, then it is converted into an executable file. Refer flow diagram here.
- Compiling a Java program creates intermediate byte code, this represents the class file.
- During execution of the program the instructions in class file is converted into assembly code by Java and is executed in the processor.
- Java Runtime Environment (JRE) is responsible for executing a java program.
- Java Virtual Machine (JVM) is responsible converting the byte code into instructions specific to operating system.
- The operating system converts the instructions into assembly code and is executed by the processor.
- JVM implementation varies based on the Operating System platform that it runs.
- One can notice during JDK download the options are provided for Linux, Mac and Windows.
- This makes java to compile once and run in any platform.

Difference between JDK, JRE and JVM

Activity

- Java Virtual Machine (JVM)
  - Open "jdk" folder in windows explorer
  - Go to bin\server
  - There is a file named jvm.dll which represents the JVM
  - JVM is a component of JRE that executes the Java program line by line
  - Operating System specific implementation
- JRE (Java Runtime Environment)
  - Open "jdk" folder in windows explorer
  - Go to bin folder
  - File "java.exe" in this folder is executed when running the "java" command in command line.
  - Contains JVM, core classes and supporting libraries
  - If somebody wants to just execute Java programs then installation of JRE alone is sufficient
- JDK (Java Development Kit)
  - Environment to develop and execute Java programs
  - Contains JRE
  - A Java developer needs JDK to develop applications using Java
  - The primary tool is the "javac.exe" file in bin folder that compiles a java source file into class file.

Questions

- Why is Java called as Platform Independent?
- JVM is common for all operating system. [True / False]
- Once class file is created it can be executed in any platform. [True / False]
- What does JVM stand for?
- What is the purpose of JVM?
- What does JRE stand for?
- What is the purpose of JRE?
- What does JDK stand for?
- What is the purpose of JDK?

Calculate Profit using primitive int and arithmetic operators

Problem

Calculate the Profit based on Buying Price and Selling Price.

Profit = Selling Price – Buying Price

The output should be printed as given below. Hard code value of buyingPrice and sellingPrice as 15 and 25 respectively.

Profit = 10

Solution

- Create new file named "CalculateProfit.java" in "java-learn" folder and implement coding below.

```
public class ProfitCalculator {
    public static void main(String args[]) {
        int buyingPrice = 15;
        int sellingPrice = 25;
        int profit = sellingPrice - buyingPrice;

        System.out.println("Profit = " + profit);
    }
}
```

- Compile and run the program

Explanation

- In the main method, following are the steps done:
  - The variable buyingPrice is assigned the value 15.
  - The variable sellingPrice is assigned the value 25.
  - Profile is calculated by subtracting buyingPrice from sellingPrice.
  - Calculated profit is printed in the screen.
- Let us understand the following line:

```
int buyingPrice = 15;
```

  - 'int' is a primitive data type in Java to define numbers.
  - 'buyingPrice' is a variable name.
  - '=' is an assignment operator.
  - '15' is an integer literal.
- Find below the list of primitive types:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Table Courtesy: https://www.w3schools.com/java/java_data_types.asp

- Variable naming conventions:

```
int buyingPrice = 15;
```

  - Variable names should be defined in full words and should not contain short forms.
  - Variables to represent index can be defined as i, j or k.
  - It should be defined in Camel Case. Find below the rules for Camel Case:
    - The first word should be defined in all lower case
    - The subsequent words should be defined with initial character in capital and rest in lower case.
    - Examples:
      - age
      - length
      - breath
      - stadiumCapacity
      - cylinderVolume
  - Variable names should not contain the data type definition.
  - The following code will compile correctly, what is the logic being implemented here:

```
int a = 24;
int b = 365;
int c = a * b;
```

  - Try to understand the code below:

```
int age = 24;
int daysInYear = 365;
int ageInDays = age * daysInYear;
```

- The assignment operator (=) helps in assigned value to a variable.

```
int buyingPrice = 15;
```

  - Before the assignment operator it should be a variable name.
  - After the assignment operator it can be a literal or another variable or expression.
  - In this example, we have used an integer literal.
  - In profit definition line the assignment operator is used to assign an arithmetic expression.

- '15' defined here is an integer literal.

```
int buyingPrice = 15;
```

  - Refer examples of other literals below:

```
int degreeCelcius = -25;           // negative literal
float price = 22.35f;              // single precision floating point
double mass = 2.7182818284590450D; // double precision floating point
double milkywayDistance = 1.0e+18; // represent exponential form.
char female = 'f';                 // represent a single character

int octalNumber = 0125;
int hexadecimalNumber = 0x1AF;

System.out.println("degreeCelcius: " + degreeCelcius);
System.out.println("price: " + price);
System.out.println("mass: " + mass);
System.out.println("milkywayDistance: " + milkywayDistance);
System.out.println("octalNumber: " + octalNumber);
System.out.println("hexadecimalNumber: " + hexadecimalNumber);

System.out.println(10 * 5 + 100 / 10 - 5 + 7 % 2);
```

- Let us now understand this line below:

```
int profit = sellingPrice - buyingPrice;
```

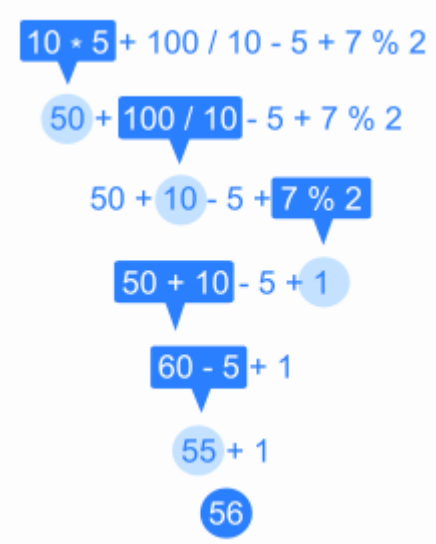
  - A variable named profit is defined with primitive data type int.
  - After assignment operator an arithmetic expression is defined.
  - Hyphen represents arithmetic subtraction.
  - Find below list of all arithmetic operators

```
+ represents addition
- represents subtraction
* represents multiplication
/ represents division
% represents modulus (remainder)
++ represents increment operator
-- represents decrement operator
```

  - Expression evaluation follows the PEMDAS standard
    - Parenthesis
    - Exponents
    - Multiplication
    - Division
    - Addition
    - Subtraction

- The following Arithmetic Expression will be evaluated as given below:

```
System.out.println(10 * 5 + 100 / 10 - 5 + 7 % 2);
```



Courtesy: https://en.wikibooks.org/wiki/Java_Programming/Arithmetic_expressions

- Let us understand the line below:

```
System.out.println("Profit = " + profit);
```

  - Specifying content within double quotes represents and String literal.
  - String is not a primitive data type; it is an in-built class.
  - The arithmetic operator here performs string concatenation instead of arithmetic addition.
  - Either if the left-hand side or right-hand side of the addition operator is a string, then it performs a string concatenation.

Exercise Problem

Based on length and breadth calculate the area of a rectangle. [Area = Length x Breadth]

Questions

- List any five primitive data types.
- What data type can be used for representing distance between planets in kilometers?
- List the naming convention standards for defining a variable name.
- How to represent the following literals?
  - Character, Float, Double, Long, Octal Number, Hexadecimal Number
- What is an assignment operator?
- What is the operator used to represent division?
- What is an arithmetic operator?
- What standard is followed when evaluating an arithmetic expression?

Calculate discount using primitive float

Problem

Calculate the Discount based on Bill Amount and Discount Percentage.

Discount = Bill Amount x Discount Percentage / 100

Example Calculation

Bill Amount = 1982

Discount Percentage = 18

Discount = 1982 x 18 / 100 = 356.76

Sample Output

Discounted Amount = 356.76

Solution

- Create new file named "CalculateDiscount.java" in "java-learn" folder
- Include a main method within this class.
- Implement the following steps in main method:
  - Create integer variable for Bill Amount and assign the value as 1982
  - Create integer variable for Discount Percentage and assign the value as 18
  - Calculate Discount using the above formula and assign to another variable integer variable for Discount
- Print the calculated discount as expected in the sample output.
- The result will truncate the fractional part of the result since int data type has been used. Change all the int data type to float data type and check the result.

Exercise Problem

Calculate Fahrenheit value based on Celsius.

$Fahrenheit = \left(Celsius \times \frac{9}{5}\right) + 32$

Example Calculation

Celsius = 12

Fahrenheit = $\left(12 \times \frac{9}{5}\right) + 32 = 53.6$

Sample Output

Fahrenheit = 53.6

Reference

- Go through Primitive Data Types section in this https://www.w3schools.com/java/java_data_types.asp

Implement discount calculation using Keyboard Input

Problem

The current discount calculation hard codes the Bill Amount and Discount Percentage values. Instead of hard coding the values get the input from keyboard prompting the user.

Solution

- Open "CalculateDiscount.java" in Notepad++
- Include the following import as the first line of the program.

```
import java.util.Scanner;
```

- Implement the code as specified below within the main method:

```
public static void main(String args[]) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter Bill Amount");
    float billAmount = scanner.nextFloat();

    System.out.println("Enter Discount Percentage");
    float discountPercentage = scanner.nextFloat();

    float discount = billAmount * discountPercentage / 100;
    System.out.println("Discount = " + discount);
}
```

- Compile and run the program.

Reference

Go through all the sections provided in the link https://www.w3schools.com/java/java_user_input.asp

Exercise Problem

Convert the Fahrenheit calculation program to get Celsius input using keyboard.

Questions

- What is the class to read input from keyboard?
- Which is the java package for Scanner class?
- What method to use when reading float value from keyboard?

Find out eligibility to vote using 'if'

Problem

Given the age of an Indian national, find out if the person is eligible to vote or not. In India, a person who is not less than 18 years has the eligibility to vote. Get the age input using Scanner.

Sample output when age less than 18 years

```
Enter the age
10
Cannot Vote
```

Sample output when age is 18 years

```
Enter the age
18
Can Vote
```

Sample output when age is not less than 18 years

```
Enter the age
25
Can Vote
```

Solution

- Create new file "VoteEligibility.java" in Notepad++
- Import Scanner
- Class name: VoteEligibility
- Code in main method:
  - Get the age using Scanner and assign to variable name "age".
  - Include the following if block to display the voting eligibility:

```
if (age >= 18) {
    System.out.println("Can Vote");
    return;
}
System.out.println("Cannot Vote");
```

- Compile and run the program.

Reference

- Go through "The if Statement" section in the link https://www.w3schools.com/java/java_conditions.asp
- Go through "Java Comparison Operators section in the link https://www.w3schools.com/java/java_operators.asp

Exercise Problem

Given the score of a student in a subject, find out if the student has passed or failed. The passing score is 35.

Sample output score is less than 35

```
Enter the score
```

```
22
Fail
```

Sample output when score is 35

```
Enter the score
35
Pass
```

Sample output when score is greater than 35

```
Enter the score
72
Pass
```

Reference

- Go through "The if Statement" section in the link https://www.w3schools.com/java/java_conditions.asp
- Go through "Java Comparison Operators section in the link https://www.w3schools.com/java/java_operators.asp

Questions

- Why do we need branching statements like if?
- Explain how "if" statement works?
- What is a relational operator?
- Which relational operator helps to check if values are not same?
- What is the resultant data type after a relational operation?

Find out eligibility to vote using 'if' and 'else'

Problem

Change the VotingEligibility class to implement using if and else.

Solution

- Open "VoteEligibility.java" in Notepad++
- Change the following code:

```
if (age >= 18) {
    System.out.println("Can Vote");
    return;
}
System.out.println("Cannot Vote");
```

- As specified below:

```
if (age >= 18) {
    System.out.println("Can Vote");
} else {
    System.out.println("Cannot Vote");
}
```

- Compile and run the program.

Reference

- Go through "The else Statement" section in the link https://www.w3schools.com/java/java_conditions.asp

Exercise Problem

Change the Pass or Fail program to use if and else.

Questions

- Explain how "if .. else" block works?

Identify average performers using and operator

Problem

Based on the score of a student, identify if he is an average performer or not. Classify as average student if the score is between 60 to 70, both values inclusive.

Sample output score is within the range

```
Enter the score
65
Average student
```

Sample output when score is lesser than the range

```
Enter the score
35
Not an average student
```

Sample output when score is greater than the range

```
Enter the score
72
Not an average student
```

Sample output when score is 60

```
Enter the score
60
Average student
```

Sample output when score is 70

```
Enter the score
70
Average student
```

Solution

- Create new file "AverageStudent.java" in Notepad++
- Class Name: AverageStudent
- Implement the following code in main method:
  - Display the message "Enter the score"
  - Get the score of the student using Scanner
  - Implement the following code:

```
if (score >= 60 && score <= 70) {
    System.out.println("Average student");
} else {
    System.out.println("Not an average student");
}
```

- Compile and run the program.

Reference

- Go through "Logical Operators" section in the link https://www.w3schools.com/java/java_operators.asp

Questions

- What are Logical Operators?
- How does Logical Operators differ from Relational Operators?
- List out the Logical Operators and it's purpose?

Classify students using if..else if and operator

Problem

Based on the score of a student, classify them as Poor, Below Average, Average, Good, Extraordinary. Find below the table that specifies how to classify based on the score.

Score Range	Classification
0 – 34	Poor
35 – 59	Below Average
60 – 75	Average
76 – 80	Above Average
81 – 100	Extraordinary

Sample output for Poor classification

```
Enter the score
32
Poor
```

Sample output for Below Average

```
Enter the score
29
Below Average
```

Sample output for Average

```
Enter the score
65
Average
```

Sample output for Above Average

```
Enter the score
78
Above Average
```

Sample output Extraordinary

```
Enter the score
92
Extraordinary
```

Solution

- Create new file "StudentClassifier.java" in Notepad++
- Class Name: StudentClassifier
- Implement the following code in main method:
  - Display the message "Enter the score"
  - Get the score of the student using Scanner
  - Implement the following code:

```
if (score >= 0 && score <= 34) {
    System.out.println("Poor");
} else if (score > 34 && score <= 59) {
    System.out.println("Below Average");
} else if (score > 60 && score <= 75) {
    System.out.println("Average");
} else if (score > 75 && score <= 80) {
    System.out.println("Above Average");
} else if (score > 80 && score <= 100) {
    System.out.println("Extraordinary");
}
```

- Compile and run the program.

Reference

- Go through "The else if statement" section in the link https://www.w3schools.com/java/java_conditions.asp

Reference for Switch statement and ternary operator

- Go through all the sections in the link https://www.w3schools.com/java/java_switch.asp
- Go through this link https://www.programiz.com/java-programming/ternary-operator to understand about ternary operator

Questions

- What is the difference between if .. else and if .. else if?
- What is the purpose of case statement in switch .. case?
- What does "?" denote in a ternary operator?

Find Web URL using method in String

Problem

Find out if a text provided is a web URL or not. If a text starts with "http://" or "https://" then it can be considered as a web URL.

Sample output for http URL

```
Enter the URL
http://www.google.com
This is a Web URL
```

Sample output for https URL

```
Enter the URL
https://www.google.com
This is a Web URL
```

Sample output for text without http or https

```
Enter the URL
google
This is not a Web URL
```

Solution

- Create new file "WebUrlChecker.java" in Notepad++
- Class Name: WebUrlChecker
- Implement the following code in main method:
  - Using Scanner get the URL and store it in a variable named "url".
  - Open JDK documentation in browser for String and find out how to use the startsWith() method.
  - Use the method and display whether it is a Web URL or not.
- Compile and run the program.

Exercise Problem

Validate whether a text input by a user is within the expected range or not. For a given Short Name find out whether it has minimum 2 characters and maximum 6 characters.

Sample output for name less than 2 characters

```
Enter Short Name
L
This is a not valid Short Name
```

Sample output for valid name

```
Enter Short Name
John
This is a valid Short Name
```

Sample output for name longer than 6 characters

```
Enter Short Name
Shankar
This is not a valid Short Name
```

Extract minutes from time using substring() method

Problem

Given a time in "hh:mm:ss" format extract the minutes. Refer sample output below for example.

Sample output 1

```
Enter Time in hh:mm:ss format
15:25:34
Minutes: 25
```

Sample output 2

```
Enter Time in hh:mm:ss format
15:03:34
Minutes: 03
```

Solution

- Create new file "TimeExtractor.java" in Notepad++
- Class Name: TimeExtractor
- Implement the following code in main method:
  - Using Scanner get the time in "hh:mm:ss" format and store it in a variable named "time".
  - In JDK documentation for String find out how to use substring() method to extract time.
  - Display the extracted result.
- Compile and run the program.

Reference

- String is immutable, which means that once a string is created it cannot be changed or altered.
- Go through the following methods in JDK documentation for string and understand the purpose of these methods:
  - toUpperCase()
  - toLowerCase()
  - contains()
  - endsWith()
  - indexOf()
  - lastIndexOf()
  - split()
  - trim()
  - valueOf()

Questions

- Why is String called immutable?
- Explain how to use the substring() function?

Managing multiple values using Array

Challenge Question

If there is a need to store scores of students in classroom of strength 45, what will be the approach to store these values as Java variables? How many variables needs to be created?

Problem

For 5 students in classroom, compute the average marks scored by the classroom in science subject. The scores of the students in science subject are 48, 55, 92, 78, 64. The formula to compute the score is to add all the scores and divide it by 5.

Sample output

```
Average score in science is: 67.5
```

Solution #1 using curly brace

-

Solution #2 using array size definition

Reference

- Go through the following sections in the link https://www.w3schools.com/java/java_arrays.asp:
  - Java Arrays
  - Access the Elements of an Array
  - Change an Array Element
  - Array Length

Questions

- What is the purpose of using arrays?
- Identify the two ways by which an array can be defined.
- Is it possible to change the value of an array item? If so, how can that be done.
- Is it possible to create an array for float values?

Find recurring TV program schedule using Loops

Problem

A TV anchor has to host a program once in 3 days from 2nd January. List the dates on which the program needs hosted during the month of January.

Sample output

```
2
5
8
11
14
17
20
23
26
29
```

Solution using 'while' loop

❖

Solution using 'while' loop and using addition assignment operator

❖

Solution using 'for' loop

❖

Calculate average classroom score using Loop

Problem

Modify the average score calculation program to dynamically get the number of students, score for each student and compute the average score.

Sample output

```
Enter the number of students:
3
1) Student Score: 80
2) Student Score: 90
3) Student Score: 100
Average Score: 90
```

Solution using 'while' loop

- ❖

Solution using 'while' loop and using addition assignment operator +=

- ❖

Solution using 'enhanced for' loop

- ❖

References

- ❖ Loop using while and do .. while - https://www.w3schools.com/java/java_while_loop.asp
- ❖ Loop using for - https://www.w3schools.com/java/java_for_loop.asp
- ❖ Break & Continue - https://www.w3schools.com/java/java_break.asp

Exercise Problem

Below is the list of documents required for personal loan. Display these items with numbering, refer sample output below. Store each item in string array, refer sample code below:

- ❖ `Driving License`
- ❖ `Passport`
- ❖ `Last 3 months Bank Statement`
- ❖ `Salary Slips of last 3 months`
- ❖ `2 Passport Size Photographs`

Store each item in string array, refer sample code below:

```
String[] days = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };
```

Sample output

```
1) Driving License
2) Passport
```

```
3) Last 3 months Bank Statement
4) Salary Slips of last 3 months
5) 2 Passport Size Photographs
```

Questions

- What is the primary objective of using loops?
- What is the difference between while and do..while loop?
- When does a while loop stop?
- What are the various elements of for loop that makes it work?
- Describe the syntax of enhanced for loop.

Print lines in bill using method

Problem

Akash is developing a billing application for a restaurant. Lines are generated using hyphen with character width of 30 characters. Refer sample bill below. The lines can be printed by using System.out.println() each time, but these lines were distracting the implementation of logic, hence Akash decides to write a method, which will print this line. Write a method for Akash that prints a single line like this. Name the method as printBillLine().

Sample output

```
------------------------------
Item        Qty     Rate    Total
------------------------------
Tea         1       15         15
Coffee      1       20         20
------------------------------
Total                          30
------------------------------
```

Solution

- Name the file as BillPrinter.java

Reference

- Reference to method - https://www.w3schools.com/java/java_methods.asp

Explanation

- Methods helps in structuring and reusing code
- Name of a method represents a verb or action that can be performed
- Naming method should be in camel case, similar to how it is done for variable names

Questions

- What is the purpose of a method?
- What are the naming conventions that needs to be followed for defining a method name?
- How to invoke a static method?

Print Bill using overloaded method

Problem

The printBillLine() method in BillPrinter by default prints 30 characters in each line. Include another method to print the line based on number of characters to be printed, which helps to print how many ever characters we want. Use this method to print the revised version of the bill print layout for displaying the line after GST.

Sample output

```
------------------------------
Item       Qty    Rate   Total
------------------------------
Tea        1      15        15
Coffee     1      20        20
------------------------------
Total                       30
------------------------------
GST 18% Inclusive
-----------------
```

Solution

- ❖

Reference

- ❖ Reference for method overloading - https://www.w3schools.com/java/java_methods_overloading.asp

Explanation

- ❖ Methods can have same name but can have difference parameters. The method to execute is picked based on the parameter.
- ❖ This feature is also called as Compile Time Polymorphism or Static Polymorphism
- ❖ Polymorphism means the condition of occurring in several different forms.

Questions

- ❖ What is method overloading?
- ❖ How is method overloading implemented?
- ❖ What are the other names for method overloading?
- ❖ What is the meaning of polymorphism?

Convert Fahrenheit calculation using method

Problem

Convert the Celsius to Fahrenheit conversion program to use method that does this conversion.

Solution

- ❖

Reference

- ❖ Reference to method - https://www.w3schools.com/java/java_methods.asp

Questions

- ❖ What is the purpose of parameters in a method?
- ❖ How are the parameters defined for a method?
- ❖ How is a method invoked with parameter?
- ❖ How does a method return data back to the calling method?

Exercise Problem

- ❖ Convert the profit calculation program to use method. The input parameters should be the buying price and selling price, the return type should be the profit.

Print account statement using Class

Problem

A bank XYZ has an account number 01302343794 has a balance of 5,000. Perform the following transactions in this account and print the Balance after each transaction.

```
Withdraw    2,000
Deposit    20,000
Deposit       500
Withdraw   15,000
Withdraw   10,000
```

Expected Output

```
Account Statement of 01302343794
Initial Balance: 5000.0

--------------------------
Type  Transaction  Balance
--------------------------
   W      2000.00  3000.00
   D     20000.00 23000.00
   D       500.00 23500.00
   W     15500.00  8000.00
   W     10000.00  8000.00
--------------------------
```

Solution (without using methods and class)

- Create class named AccountManager
- Include following steps in main() method.
- Define variables for account number and balance. Then print the result:

```
String accountNumber = "01302343794";
double balance = 5000;

System.out.println("Account Statement of " + accountNumber);
System.out.println("Initial Balance: " + balance);

System.out.println("--------------------------");
System.out.println("Type  Transaction  Balance");
System.out.println("--------------------------");
```

- Define transaction amount, perform balance calculation and print the transaction.

```
double transactionAmount = 2000;
balance = balance - transactionAmount;
System.out.printf("   W %12.2f %8.2f\n", transactionAmount, balance);
```

- Repeat the above lines for each transaction.
- Do not forget to change the "W" within the printed string to "D" for deposit transaction.
- Print dotted line using System.out.println().

Explanation

- The method printf() helps to print formatted output.
- The text %12.2f represents displaying transactionAmount with 12 decimal places and 2 fractional spaces. Similarly, %8.2f represents the formatting for balance.

Solution (using methods)

- Open AccountManager.java in Notepad++
- Create a new method calculateBalance() with the below method signature:

```
public static double calculateBalance(char transactionType,
        double transactionAmount, double balance) {
```

- Implement the following steps in the calculateBalance() method.
  - For transaction type 'D' add transaction amount with balance:

```
if (transactionType == 'D') {
    balance = balance + transactionAmount;
}
```

  - For transaction type 'S' subtract transaction amount from balance, if insufficient balance then balance is not modified:

```
if (transactionType == 'W') {
    double tempBalance = balance - transactionAmount;
    if (tempBalance >= 0) {
        balance = balance - transactionAmount;
    }
}
```

  - Print the transaction statement and return the balance. Refer code below:

```
System.out.printf("%4c %12.2f %8.2f\n", transactionType,
    transactionAmount, balance);
return balance;
```

- Make following changes in the main() method:
  - Apply multi line comment to all lines that does the transaction calculation.

```
/*transactionAmount = 20000;
balance = balance + transactionAmount;
System.out.printf("   D %12.2f %8.2f\n", transactionAmount, balance);

.

.
transactionAmount = 10000;
balance = balance - transactionAmount;
System.out.printf("   W %12.2f %8.2f\n", transactionAmount, balance);*/
```

  - Invoke calculateBalance() method for each transaction. Refer code for one transaction, implement for other transactions in a similar fashion.

```
balance = calculateBalance('W', 2000, balance);
```

- Compile and run the program to check if similar results are obtained.

Solution (using Class)

Transaction

- Create a new file named Transaction.java with class Transaction
- Include private instance variables for transaction type and transaction amount within the class as specified below:

```
public class Transaction {
    private char type;
    private double amount;
}
```

- Include a constructor for Transaction with two parameters:

```
public class Transaction {
    private char type;
    private double amount;

    public Transaction(char type, double amount) {
        this.type = type;
        this.amount = amount;
    }
}
```

- Include method to display transaction data. Since balance is not part of transaction class, it is fed as input parameter.

```
public void displayTransaction(double balance) {
    System.out.printf("%4c %12.2f %8.2f\n", type, amount, balance);
}
```

- Include testing of Transaction class using a main() method within the Transaction class. Implement the following code within main() method and execute.

```
Transaction transaction = new Transaction('W', 2000);
transaction.displayTransaction(5000);
```

- Create similar variable for each transaction with variables naming transaction2, transaction3, transaction4 and transaction5. Then print the transaction data using displayTransaction() method.
- The transaction instances creation can also be created using arrays. Comment out all the lines in the main method and include the below code:

```
Transaction[] transactions = {
    new Transaction('W', 2000),
    new Transaction('D', 20000),
    new Transaction('D', 500),
    new Transaction('W', 15000),
    new Transaction('W', 10000)
};

for (Transaction transaction: transactions) {
    transaction.displayTransaction(5000);
}
```

- Compile and run Transaction.java to check if all the transactions are displayed.

Account

- Create a new file called Account.java with class Account.
- Include below instance variables in Account class

```
public class Account {
    private String accountNumber;
    private double balance;
    private Transaction[] transactions;
}
```

- Include below constructor for this class:

```
public Account(String accountNumber, double balance,
        Transaction[] transactions) {
    this.accountNumber = accountNumber;
    this.balance = balance;
    this.transactions = transactions;
}
```

- Include deposit method to handle deposit transaction:

```
public void deposit(double transactionAmount) {
    balance += transactionAmount;
}
```

- Include withdraw method to handle withdraw transaction. Copy the code from calculateBalance() method of AccountManager.java

```
public void withdraw(double transactionAmount) {
    double tentativeBalance = balance - transactionAmount;
    if (tentativeBalance > 0) {
        this.balance = tentativeBalance;
    }
}
```

- Include a method to print the account statement. Copy code from AccountManager.java

```
public void printStatement() {
    System.out.println("Account Statement of " + accountNumber);
    System.out.println("Initial Balance: " + balance);
    System.out.println();
    System.out.println("--------------------------");
    System.out.println("Type  Transaction  Balance");
    System.out.println("--------------------------");

    for (Transaction transaction : transactions) {
        if (transaction.getType() == 'W') {
            this.withdraw(transaction.getAmount());
        } else if (transaction.getType() == 'D') {
            this.deposit(transaction.getAmount());
        }
        transaction.displayTransaction(this.balance);
    }
    System.out.println("--------------------------");
}
```

Include appropriate getType() method in Transaction class that returns the transaction amount.

- Include main method with following code to display transaction. Copy code from main method of Transaction.java for creating transaction array.

```
public static void main(String args[]) {
    Transaction[] transactions = {
        new Transaction('W', 2000),
        new Transaction('D', 20000),
        new Transaction('D', 500),
        new Transaction('W', 15000),
        new Transaction('W', 10000)
    };
    Account account = new Account("01302343794", 5000, transactions);
    account.printStatement();
}
```

- Compile and run Account.java and check if transactions are printed.

Explanation

- Provide explanation for the following:
  - Instance variables in a class
  - Constructor of a class
  - Object instance reference
  - Invoking an instance method
  - Difference between invoking instance method and static method

References

- Classes and Objects - https://www.geeksforgeeks.org/classes-objects-java/
- Encapsulation - https://www.w3schools.com/java/java_encapsulation.asp

Install Eclipse

- In browser go to https://www.eclipse.org/
- Click “Download”.
- Click “Download 64 bit”.
- This will download an executable installer file.
- Execute this program after download.
- On the window that appears select “Eclipse IDE for Enterprise Java Developers”
- Ensure Java 1.8+ VM has the JDK 13 folder that had been installed earlier.
- Select an appropriate installation folder.
- Ensure “Create start menu entry” and “Create desktop shortcut” boxes are checked.
- Click “INSTALL” to initiate the installation. This will take a while, wait until the installation completes.

Setting up the project in Eclipse

- Launch Eclipse from Start Menu
- Select a Workspace folder where all the projects of Eclipse will be stored and click Launch
- In Menu select File > New > "Project..."
- Select "Java Project" and click "Next"
- Enter the project name as "XYZBank" and click "Next"
- Click "Finish"
- Open Windows File Explorer and go to the folder where Account.java and Transaction.java files are present. Copy both the files.
- Go to Eclipse
- Expand the new project in "Project Explorer"
- Right click on "src" and select "Paste"
- Run Account.java, by right clicking the file in Project Explorer and selecting "Run As" > "Java Application"
- Check the result of the program in "Console" window.

Executing the program in Debug Mode

- Open Eclipse
- Double click on Account.java and open it in the right-hand side Text Editor
- Right click on the line number of the first line of main() method and select "Toggle Breakpoint".
- Right click on "Account.java" in "Project Explorer" and select "Debug As" > "Java Application"
- Accept the layout change for debugging
- Use "Step In" and "Step Over" option to explain how program execution works.

Viewing the runtime memory using Visual VM

- In browser go to https://visualvm.github.io/
- Click on download and extract the zip to a convenient folder.
- Open Eclipse
- Remove the breakpoint in the first line of main method using "Toggle Breakpoint" option
- Include a new breakpoint in the last line of code in main() method of Account.java
- Execute Account.java in debug mode
- Launch Visual VM by executing visualvm.exe file in bin folder of the Visual VM zip extraction
- In Visual VM, double click on "Account" is the "Applications" section in the left-hand side, which opens tabs in the right-hand side.
- Select "Sample" in this tab.
- Click "Memory"
- Click "Heap Dump"
- Click "Summary" and select "Objects"
- In the table displayed below click on "Name" column to sort based on class Name.
- Expand "Account" and navigate through the data available in this object instance.

Activity (Object reference assignment)

- Open main() method of Transaction class
- Include lines below:

```
Transaction transaction1 = new Transaction('W', 2000);
Transaction transaction2 = new Transaction('D', 3000);
transaction1 = transaction2;
transaction1.displayTransaction(5000);
```

- Include a breakpoint in the last line of this code and analyze the data in Visual VM.

Exercise Problem

Define class model about a train and the stations that it passes through. Class model details:

Train	Station	Schedule
Number Name Schedules	Code Name	Sequence Number Station Arrival Time Departure Time

Expected Output for Train 12213

```
Yasvantpur Delhi Sarai Rohilla AC Duronto Express (12213)
-------------------------------------
#  Code  Station          Arr.   Dep.
-------------------------------------
1  YPR   Yasvantpur Jn           23.40
2  GTL   Guntakal Jn      03.45  03.50
3  SC    Secundrabad Jn   08.55  09.10
4  BPQ   Balharshah       13.30  13.35
5  HBJ   Habibganj        21.20  21.25
6  JHS   Jhansi Jn        01.15  01.20
7  DEE   Delhi S Rohilla  07.00
-------------------------------------
```

Questions

- What are instance variables?
- What is a constructor?
- What is the purpose of a constructor?
- Can a class have more than one constructor?
- What is the use of “this” keyword?
- What is an instance method?
- What happens when one object instance variable is assigned to another variable?
- What is Encapsulation / Data Hiding?
- What is the effect of assigning an instance variable as public? What is the impact?
- How to create a new instance of a class?

Modify the statement to display Bank Name

Problem

Modify the account statement to include name of the bank. Refer code below.

Expected Output

```
XYZ Bank
Account Statement of 01302343794
Initial Balance: 5000.0

--------------------------
Type  Transaction  Balance
--------------------------
   W      2000.00  3000.00
   D     20000.00 23000.00
   D       500.00 23500.00
   W     15500.00  8500.00
   D     10000.00 18650.00
--------------------------
```

Solution

- Include static variable for bank name in Account class.

```
private static String bankName = "XYZ Bank";
```

- Modify the displayStatement() method to display bank name in the first line.

```
public void printStatement() {
    System.out.println(bankName);
```

Explanation

- Static variables are not initialized for each instance.
- The same static variable is used by all instances.
- Create a heap dump in Visual VM to check how static variables are created

Cash Back offer for deposit

Problem

The XYZ Bank is a new bank in the market. Currently it only operates Savings Bank accounts. To improve deposits in Savings Account, it offers cash back of 150 for a transaction deposit of 10,000. The cash back will be deposited directly into the account. Let us assume account number 01302343794 has a balance of 5,000. Find below the transactions listed. The last transaction with 10,000 deposit will get a cash back of 150, which can be found in the last line transaction in the Expected Output section below.

```
Withdraw    2,000
Deposit    20,000
Deposit       500
Withdraw   15,000
Deposit    10,000
```

Expected Output

```
Account Statement of 01302343794
Initial Balance: 5000.0

-------------------------
Type  Transaction  Balance
-------------------------
   W      2000.00  3000.00
   D     20000.00 23000.00
   D       500.00 23500.00
   W     15500.00  8500.00
   D     10000.00 18650.00
-------------------------
```

Solution

- Modify the deposit() method in Account class to add the cashback for the transaction with amount 10,000.

XYZ Banks implements Credit Cards (Inheritance)

Problem

As XYZ Bank grows, they have launched Credit Card feature. To promote the credit card usage, they offer cash back of 100 for purchases that are 5000 and above. Find below transactions for card number 5544 3322 1100 9988 with a credit balance of 20,000. In the statement W represents purchase made in the card and D denotes Card Repayment. Organize the new classes in a package named "com.xyzbank.model".

```
Purchase    2,000
Repayment   2,000
Purchase    5,000
```

Expected Output

```
Account Statement of Credit Card 5544 3322 1100 9988
Initial Balance: 20,000.00

--------------------------
Type  Transaction   Balance
--------------------------
   W      2000.00  18000.00
   D      2000.00  20000.00
   W      5000.00  15100.00
--------------------------
```

Solution (without Inheritance)

- Include new instance variable name "type" of datatype "char" in Account class. This is to differentiate an account between Savings and Credit card. This will hold the value 'S' for Savings Account and 'C' for Credit Card
- Include "type" as input parameter in the constructor.
- In withdraw() method, check if the account type is 'C', then include the logic to add cash back of 100.
- Comment out the existing transaction array definition and include transaction details as specified in this problem statement.
- Compile and run the program to check if the expected results are obtained.

Limitations of implementing without Inheritance

- Understand the complexity that will arise in this code when following features needs to be implemented:
  - Implementing Fixed Deposit
  - Implementing Recurring Deposit
  - Including a point system in Credit Card
  - Handling late payment charges in Credit Card

Solution (with Inheritance)

- Open Eclipse and expand the "XYZBank" project
- Right click on "src" select "New" and the select "Package"
- Name the package as "com.xyzbank.model"

- Copy the files Account.java and Transaction.java to this new package created in previous step.
- Create two new classes named SavingsAccount and CreditCardAccount in the package "com.xyzbank.model"
- ***Account.java***
  - Remove instance variable "type"
  - Modify the constructor to remove type as parameter and remove respective assignment within the constructor.
  - Change the modifier for accountNumber, balance and transactions as "protected".
  - Comment out the handling of credit card in withdraw() method.

```
public void withdraw(double transactionAmount) {
    double tentativeBalance = balance - transactionAmount;
    if (tentativeBalance > 0) {
        balance = tentativeBalance;
        /*if (type == 'C' & transactionAmount > 5000) {
            balance = balance + 100;
        }*/
    }
}
```

- ***SavingsAccount.java***
  - Inherit this class from Account

```
public class SavingsAccount extends Account {
```

  - Include a constructor with exactly the same set of parameters as in Account.java

```
public SavingsAccount(String accountNumber, double balance,
        Transaction[] transactions) {
    super(accountNumber, balance, transactions);
}
```

  - Include deposit() method with code as specified below:

```
public void deposit(double transactionAmount) {
    super.deposit(transactionAmount);
    if (transactionAmount == 10000) {
        balance += 150;
    }
}
```

  - Include main() method to test the SavingsAccount.java class

```
public static void main(String args[]) {
    Transaction[] transactions = {
        new Transaction('W', 2000), new Transaction('D', 20000),
        new Transaction('D', 500), new Transaction('W', 15000),
        new Transaction('D', 10000)
    };
    SavingsAccount account = new SavingsAccount("01302343794",
            5000, transactions);
    account.printStatement();
}
```

  - Compile and run the program.
- ***CreditCardAccount.java***
  - Inherit this class from Account

```
public class CreditCardAccount extends Account {
```

  - Include a constructor similar to how we implemented in SavingsAccount class.
  - Implement withdraw() method as specified below:

```
public void withdraw(double transactionAmount) {
    double tentativeBalance = balance - transactionAmount;
    if (tentativeBalance > 0) {
        balance = tentativeBalance;
        if (transactionAmount >= 5000) {
            balance = balance + 100;
        }
    }
}
```

- Implement the main() method as specified below to test CreditCardAccount class

```
public static void main(String args[]) {
    Transaction[] transactions = {
            new Transaction('W', 2000),
            new Transaction('D', 2000),
            new Transaction('W', 5000)
    };
    CreditCardAccount account =
            new CreditCardAccount("5544 3322 1100 9988",
                    20000, transactions);
    account.printStatement();
}
```

- Compile and run the program.

❖ Modify main() method in Account.java to implement runtime polymorphism:

```
public static void main(String[] args) {
    Transaction[] savingsTransactions = {
        new Transaction('W', 2000), new Transaction('D', 20000),
        new Transaction('D', 500), new Transaction('W', 15000),
        new Transaction('D', 10000)
    };
    Transaction[] cardTransactions = {
        new Transaction('W', 2000), new Transaction('D', 2000),
        new Transaction('W', 5000)
    };
    Account savingsAccount = new SavingsAccount("01302343794", 5000, savingsTransactions);
    Account cardAccount = new CreditCardAccount("5544 3322 1100 9988",
            20000, cardTransactions);
    Account[] accounts = { savingsAccount, cardAccount };
    for (Account account : accounts) {
        account.printStatement();
    }
}
```

❖ ***Account.java***

- In this given scenario there is no need for creating an instance of Account class.
- This class can be safely converted into abstract class.

```
public abstract class Account {
```

- Run Account class and check if it works fine after the above change.

Explanation for package

❖ Packages helps group related classes

❖ Package naming convention:

- Package name should be formed with reversal of the domain name.
- Let us take Google as an example.
- A java project in google should start with “com.google” as package, since their website URL ends with google.com.

- The package name should be followed with the name of the project, if the project is for Chrome browser, the package name should be "com.google.chrome".
- Package name should be followed by modules within the Chrome project. For example, coding related to the menu in the Chrome browser, then the classes for menu can be under the package "com.google.chrome.menu".

❖ In the previous example, for the XYZ Bank we have used the package name as "com.xyzbank".

❖ Import is not required if the reference class resides in the same package.

❖ Going forward, let us define package name as "com.company".

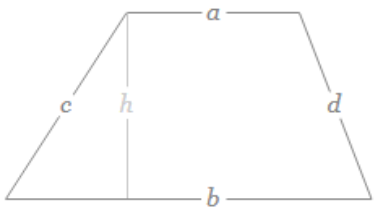
❖ In the following examples "com.company" will be used as package prefix, but change it with your name instead.

Exercise Problem

❖ Implement area calculation for the following Quadrilaterals:

- Rectangle (Area = Height x Width)
- Square (Area = Side x Side)
- Trapezium (Area formula below)

$$A = \frac{a+b}{2}h$$



❖ Hints

- Define the package name as "com.company.inheritance"
- Define Quadrilateral as an abstract class with height and width as attributes. Implement area() method which multiplies height and width.
- Extend 3 classes Rectangle, Square and Trapezium from Quadrilateral.
- The extended classes should reuse the area() method without implementing it in the child class.

Reference

❖ Inheritance - https://www.geeksforgeeks.org/inheritance-in-java/

❖ Runtime Polymorphism - https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/

❖ Super Keyword - https://www.geeksforgeeks.org/super-keyword/

❖ Access Modifiers - https://www.geeksforgeeks.org/access-modifiers-java/

Questions

❖ What is inheritance?

- How to implement inheritance?
- How to make instance variables accessible to child class?
- How to refer constructors, instance variables and methods of parent class from child class?
- What is runtime polymorphism?
- How is runtime polymorphism implemented in Java?

Implement Bike using Interface

Problem

Refer below bike details. Implement a behavior model for representation of the functions of a bike.

	Hero Splendor	Harley Davidson Fat Boy
		
No. of Gears	4	6
Top Speed	80 Kmph	175 Kmph

Following are the common behavior that is expected in any bike:

- Start or Stop the bike
- Increase or Decrease Gear
- Accelerate
- Brake

Expected Output

```
Splendor running at 10Km/h on gear 1.   Fat Boy running at 30Km/h on gear 1.
Splendor running at 20Km/h on gear 2.   Fat Boy running at 60Km/h on gear 2.
Splendor running at 30Km/h on gear 3.   Fat Boy running at 90Km/h on gear 3.
Splendor running at 40Km/h on gear 4.   Fat Boy running at 120Km/h on gear 4.
Splendor running at 50Km/h on gear 4.   Fat Boy running at 150Km/h on gear 5.
Splendor running at 60Km/h on gear 4.   Fat Boy running at 175Km/h on gear 6.
Splendor running at 70Km/h on gear 4.   Fat Boy running at 175Km/h on gear 6.
Splendor running at 80Km/h on gear 4.   Fat Boy running at 175Km/h on gear 6.
Splendor running at 80Km/h on gear 4.   Fat Boy running at 175Km/h on gear 6.
Splendor running at 80Km/h on gear 4.   Fat Boy running at 175Km/h on gear 6.
```

Solution

Bike.java

- Create new package "com.company.abstraction"
- Right click on this new package > New > Interface
- Name the interface as Bike
- Include the following methods within the interface:

```
public interface Bike {

    void start();
    void stop();
    void increaseGear();
    void decreaseGear();
    void accelerate();
    void brake();

}
```

HeroSplendor.java

- Create new class com.company.abstraction.HeroSplendor which implements Bike interface.

```
public class HeroSplendor implements Bike {
```

- Define following static and instance variables:

```
private static int TOP_GEAR = 4;
private static int TOP_SPEED = 80;

private int speed;
private int gear;
```

- Implement start() and stop() method as specified below:

```
public void start() {
    speed = 0;
    gear = 0;
}

public void stop() {
    speed = 0;
}
```

- Implement gear increase and decrease methods as specified below:

```
public void increaseGear() {
    gear++;
    if (gear > TOP_GEAR) {
        gear = TOP_GEAR;
    }
}

public void decreaseGear() {
    gear--;
    if (gear < 0) {
        gear = 0;
    }
}
```

- Implement accelerate and brake method as specified below:

```
public void accelerate() {
    speed += 10;
    if (speed > TOP_SPEED) {
        speed = TOP_SPEED;
    }
}

public void brake() {
    speed -= 10;
    if (speed < 0) {
        speed = 0;
    }
}
```

- To display the current status of a bike implement the toString() method as specified below:

```
public String toString() {
    return "Splendor running at " + speed + "Km/h on gear " + gear + ".";
}
```

HarleyDavidsonFatBoy.java

- Create class com.company.abstraction.FatBoy that implements Bike interface
- Copy all the implementation from HeroSplendor class and paste it.
- Change TOP_GEAR value to 6.
- Change TOP_SPEED value to 175.

BikeTester.java

- Create class com.company.abstraction.BikeTester
- Include main() method
- Create an instance of HeroSplendor and HarleyDavidsonFatBoy, start the bikes and increase the acceleration in a loop. Refer code below:

```
public class BikeTester {

    public static void main(String args[]) {
        Bike splendor = new HeroSplendor();
        Bike fatBoy = new HarleyDavidsonFatBoy();

        splendor.start();
        fatBoy.start();

        for (int i = 0; i < 10; i++) {
            splendor.increaseGear();
            splendor.accelerate();

            fatBoy.increaseGear();
            fatBoy.accelerate();

            System.out.println(splendor + "    " + fatBoy);
        }

        splendor.stop();
        fatBoy.stop();
    }

}
```

- Run BikeTest code and check if the results appear as expected.

Explanation for interface

- Interface defines the behavior of an entity.
- Interface does not define how the behavior should be. It is the responsibility of the implementing class to define the behavior.
- Use interface keyword to define an interface.
- Define method signatures in interface without method implementation.
- The implementing class uses “implements” keyword to identify the interface that needs to be implemented.
- More interfaces can be specified by separating it with comma.
- Implementing an interface forces to the class to provide an implementation for each method that is defined in the interface. If not implemented the compilation fails.
- A reference of an interface can be assigned to any one of its implemented class instance. This interface reference can be used to invoke all the methods defined within the interface.
- Interface acts as an abstracted contract between a specification and actual implementation.
- Examples of interface that is used as contract between multiple implementations:
    - There are multiple java based web servers available viz., Tomcat, GlassFish, WebSphere, WebLogic, JBoss.
    - All these servers implements the standard specification of Servlet and JSP, which are defined as part of Java Community Process.
    - This makes it easier to deploy a web application in any one of the above specified web servers seamlessly.

- Another example is that JDBC specification helps to define the standard interface for connecting to any database, which makes it easier to switch to any database without having the need to change the coding already done.

Explanation for toString() method

- The ultimate parent of any class is java.lang.Object
- In Project Explorer window, right click on SavingsAccount and select "Open Type Hierarchy"
- This displays a new window.
- Notice that parent of SavingsAccount is Account and the parent of Account is Object.
- When defining Account class we have not defined extends, java implicitly assigns java.lang.Object as the parent class.
- Double click Object in the Type Hierarchy window. This displays the Object class code in the editor window and displays the methods of Object class in the bottom window.
- Click the toString() method and refer the code. This method displays the class name and Hash Code. This is the default implementation of toString() in the parent Object class.
- Comment out the toString() method in HeroSplendor.java and run the BikerTest program, then check the result, which will match the implementation of toString() method in Object class.
- If toString() method is implemented in the child class, then through runtime polymorphism child classes method is executed.
- Not let us explore the println() method implementation of PrintStream class.
- Expand "JRE System Library" under the java-learn project
- Expand "java.base" > "java.io" > "PrintStream.class" > "PrintStream"
- Double click on println(Object)
- Click valueOf() method and Ctrl key in parallel
- In the valueOf() method implementation the toString() method is invoked which calls the toString() method implemented in HeroSplendor. If this toString() method is not implemented then the default implementation in parent class Object is executed.=

Reference

- Interface - https://www.w3schools.com/java/java_interface.asp
- Object class - https://www.geeksforgeeks.org/object-class-in-java/
- Difference between Interface and Abstract class - https://www.javatpoint.com/difference-between-abstract-class-and-interface

Questions

- What is an interface in Java?
- How to define an interface?
- How to implement an interface?
- What is the purpose of using interface?
- What is Object class?
- What is the significance of toString() method in Object class?

- List out the differences between an interface and Abstract class

Include date and time of transaction in Bank Statement using LocalDateTime

Problem

Modify the bank statement to display date and time of each transaction. The input for the date field is in "dd/mm/yy hh:mm" format. The display of the date in the account statement should be as specified below in "dd mmm yy hh:mm" format. Refer sample below:

Expected Output

```
Account Statement of 01302343794
Initial Balance: 5000.0

-----------------------------------------
Date              Type  Transaction  Balance
-----------------------------------------
01 Jan 19 07:25     W      2000.00  3000.00
08 Feb 19 10:42     D     20000.00 23000.00
08 Feb 19 12:03     D       500.00 23500.00
19 Mar 19 15:48     W     15500.00  8500.00
15 Apr 19 23:25     D     10000.00 18650.00
-----------------------------------------
```

Solution

- Open com.xyzbank.model.Transaction class in Eclipse
- Include the following imports

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
```

- Include formatter for parsing the input date:

```
private static DateTimeFormatter INPUT_DATE_FORMAT =
        DateTimeFormatter.ofPattern("dd/MM/yy HH:mm");
```

- Include formatter for formatting date in the account statement:

```
private static DateTimeFormatter STATEMENT_DATE_FORMAT =
        DateTimeFormatter.ofPattern("dd MMM yy HH:mm");
```

- Include instance variable named "date" of type "java.time.LocalDateTime":

```
private LocalDateTime date;
```

- Modify constructor to include this parameter and set appropriate value. Parse the input string for date in "dd/mm/yy hh:mm" format to LocalDateTime object.

```
public Transaction(String date, char type, double amount) {
    LocalDateTime localDateTime = LocalDateTime.parse(date, INPUT_DATE_FORMAT);
    this.date = localDateTime;
    this.type = type;
    this.amount = amount;
}
```

- Modify displayTransaction() method to print the date is a specified format.

```
public void displayTransaction(double balance) {
    String formattedDate = date.format(STATEMENT_DATE_FORMAT);
    System.out.printf("%15s %5c %12.2f %8.2f\n", formattedDate, type, amount, balance);
}
```

- In main() method, modify the Transaction instance creation with date

```
public static void main(String[] args) {
    Transaction transaction = new Transaction("01/01/19 15:43", 'W', 2000);
    transaction.displayTransaction(5000);
}
```

- Run this class and test if the date is displayed
- Modify transaction instances creation in SavingsAccount.java within main() method.

```
Transaction[] transactions = {
    new Transaction("01/01/19 07:25", 'W', 2000),
    new Transaction("08/02/19 10:42", 'D', 20000),
    new Transaction("08/02/19 12:03", 'D', 500),
    new Transaction("19/03/19 15:48", 'W', 15000),
    new Transaction("15/04/19 23:25", 'D', 10000)
};
```

- Compile and run SavingsAccount.java
- Similarly, modify transaction instances creation in CreditCardAccount.java and Account.java.
- In Account.java modify the printStatement() method to include spacing and line width.
- Compile and run the program to check if date for each transaction is printed. Refer sample output below:

```
XYZ Bank
Account Statement of 01302343794
Initial Balance: 5000.0
From: 01 Jan 19 00:00
To: 28 Apr 19 00:00
-----------------------------------------
Date             Type  Transaction  Balance
-----------------------------------------
01 Jan 19 07:25     W      2000.00  3000.00
08 Feb 19 10:42     D     20000.00 23000.00
08 Feb 19 12:03     D       500.00 23500.00
19 Mar 19 15:48     W     15000.00  8500.00
15 Apr 19 23:25     D     10000.00 18650.00
-----------------------------------------

XYZ Bank
Account Statement of 5544 3322 1100 9988
Initial Balance: 20000.0
From: 01 Jan 19 00:00
To: 28 Apr 19 00:00
-----------------------------------------
Date             Type  Transaction  Balance
-----------------------------------------
01 Jan 19 07:25     W      2000.00 18000.00
08 Feb 19 10:42     D      2000.00 20000.00
08 Feb 19 12:03     W      5000.00 15100.00
-----------------------------------------
```

Explanation

- LocalDateTime is a class in java.time package to handle date and time data
- The java.time is a new package introduced as part of JDK 1.8 to handle the drawbacks of legacy classes java.util.Date, java.util.Calendar and java.util.TimeZone.
- DateTimeFormatter class in java.time.format helps in formatting or parsing the data for date stored in LocalDateTime class.
- The ofPattern() method in formatter class helps in defining a custom date format.
- The parse() method in LocalDateTime class helps in converting a string to LocalDateTime
- The format() method in LocalDateTime class helps in converting LocalDateTime to a custom string format.

References

- Legacy Date Library comparison - https://docs.oracle.com/javase/tutorial/datetime/iso/legacy.html
- Date Time formatting example - https://www.logicbig.com/how-to/code-snippets/jcode-java-8-date-time-api-localdatetime-format.html
- Date Time formatting API Reference - https://docs.oracle.com/javase/8/docs/api/index.html?java/time/format/DateTimeFormatter.html
- LocalDate - https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html
- LocalDateTime - https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html
- LocalTime - https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html
- DateTimeFormatter - https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

Exercise Problem #1

A news website wants to display the time when an article is published in the format given below. Write a program to accept the date and time details and print it in the expected format.

```
Mar 26, 2020, 03.46 PM
```

`Package: com.company.date`

Sample Output

```
Enter Date in dd/mm/yy hh:mm format:
12/12/12 23:25
Dec 12, 2012, 23.25 pm
```

Exercise Problem #2

A software is being developed for a training institute. Given a training batch start date and the number of weeks the course duration is, find the end date of the training.

`Package: com.company.date`

Sample Output

```
Enter course start date in dd/mm/yy format:
03/03/19
Enter duration of batch in weeks:
3
Training ends on 24/03/19
```

Display transactions that falls within two specified dates

Problem

Modify the bank statement to display date and time of each transaction. Refer sample below that displays all the transactions during the months Jan and Feb of year 2019:

Expected Output

```
XYZ Bank
Account Statement of 01302343794
Initial Balance: 5000.0
From: 01 Jan 19 00:00
To: 28 Apr 19 00:00
-----------------------------------------
Date             Type  Transaction  Balance
-----------------------------------------
01 Jan 19 07:25     W      2000.00  3000.00
08 Feb 19 10:42     D     20000.00 23000.00
08 Feb 19 12:03     D       500.00 23500.00
-----------------------------------------

XYZ Bank
Account Statement of 5544 3322 1100 9988
Initial Balance: 20000.0
From: 01 Jan 19 00:00
To: 28 Apr 19 23:59
-----------------------------------------
Date             Type  Transaction  Balance
-----------------------------------------
01 Jan 19 07:25     W      2000.00 18000.00
08 Feb 19 10:42     D      2000.00 20000.00
08 Feb 19 12:03     W      5000.00 15100.00
-----------------------------------------
```

Solution

- Open com.xyzbank.model.Account class in Eclipse
- Include the following imports

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
```

- Include formatter for formatting date in the account statement:

```
private static DateTimeFormatter STATEMENT_DATE_FORMAT =
        DateTimeFormatter.ofPattern("dd MMM yy HH:mm");
```

- Modify printStatement() method to include parameters for start date and end date:

```
public void printStatement(LocalDateTime startDate, LocalDateTime endDate) {
```

- Include the below statements in printStatement() method to display the start and end dates after the printing of the balance:

```
System.out.println("From: " + startDate.format(STATEMENT_DATE_FORMAT));
System.out.println("To: " + endDate.format(STATEMENT_DATE_FORMAT));
```

- Modify the for loop in printStatement() as specified below.

```
        for (Transaction transaction : transactions) {
            char type = transaction.getType();
            double amount = transaction.getAmount();
            LocalDateTime date = transaction.getDate();

            if (date != null && date.isAfter(startDate) && date.isBefore(endDate)) {
                transaction.displayTransaction(this.balance);
            }

            if (type == 'W') {
                this.withdraw(amount);
            } else if (type == 'D') {
                this.deposit(amount);
            }
        }
```

- Modify the invocation of printStatement() method in main() method to provide the start and end date:

```
for (Account account : accounts) {
    account.printStatement(LocalDateTime.of(2019, 1, 1, 0, 0),
            LocalDateTime.of(2019, 4, 28, 23, 59));
    System.out.println();
}
```

- Compile and run Account.java to check if only filtered transactions are displayed.

Explanation

- The methods isAfter() and isBefore() methods in LocalDateTime class helps to check if a transaction falls within start date and end date.

Reference

- LocalDateTime - https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html

Compute age using Period

Problem

Given the date of birth find the age in years.

Expected Output

```
Enter Date of Birth in dd/mm/yyyy format
15/04/1980
Age: 39
```

Solution

- Create class named "com.company.date.AgeCalculator"
- Import java.util.Scanner
- main()
  - Accept the date as String using Scanner

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter Date of Birth in dd/mm/yyyy format");
String input = scanner.nextLine();
```

  - Create a formatter and parse the date to convert into LocalDate.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
LocalDate date = LocalDate.parse(input, formatter);
```

  - Using Period find out the age.

```
Period period = Period.between(date, LocalDate.now());
int age = period.getYears();
System.out.println("Age: " + age);
```

- Compile and run the program.

Explanation

- Period class helps in identifying years, months, weeks and days between two LocalDate
- Duration class helps in identifying hours, minutes and seconds between two LocalDateTime

Reference

- Period API docs - https://docs.oracle.com/javase/8/docs/api/java/time/Period.html
- Duration API docs - https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html
- Period and Duration examples - https://mkyong.com/java8/java-8-period-and-duration-examples/

Find a citizen using equals() method

Problem

Given the details about a list of citizens, find out a citizen based on National ID, First Name, Last Name and Date of Birth. Following are the details stored about a citizen of a country.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Riki	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

Expected Output (when citizen match found)

```
Enter National ID
987977
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen FOUND
Citizen [nationalId=987977, firstName=John, lastName=Sam, gender=M, dateOfBirth=2000-10-17]
```

Expected Output (when citizen match not found)

```
Enter National ID
985765
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen NOT FOUND
```

Solution (without using equals())

Citizen.java

- Create package com.company.equals
- Create class Citizen in this new package.
- Include following instance variables:

```
private int nationalId;
private String firstName;
private String lastName;
private char gender;
private LocalDate dateOfBirth;
```

- Place the cursor after the instance variables
- Select Source > Generate constructor using fields…
- Uncheck "gender" and click "Generate"
- Place the cursor after the instance variables
- Select Source > Generate constructor using fields…
- Ensure all the instance variables are check
- Click "Generate"
- Place the cursor after the 2nd constructor
- Select Source > Generate toString()…
- Ensure all the instance variables are checked
- In "Code Style" select "String.format/MessageFormat"
- Click "Generate"

CitizenDataManager.java

- Create new class com.company.equals.CitizenDataManager
- Copy and paste the code below which generates the sample data of Citizen array:

```
package com.company.equals;

import java.time.LocalDate;

public class CitizenDataManager {

    public static Citizen[] getCitizenData() {
        Citizen citizen1 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));
        Citizen citizen2 = new Citizen(223463, "Jack", "Potter", 'M', LocalDate.of(1996, 12, 18));
        Citizen citizen3 = new Citizen(878237, "Lily", "Charlie", 'F', LocalDate.of(2001, 4, 7));
        Citizen citizen4 = new Citizen(233467, "Linda", "Johnson", 'F', LocalDate.of(1974, 11, 1));
        Citizen citizen5 = new Citizen(765657, "Riki", "Elliott", 'F', LocalDate.of(1999, 9, 23));
        Citizen citizen6 = new Citizen(987977, "John", "Sam", 'M', LocalDate.of(2000, 10, 17));

        Citizen[] citizen = {citizen1, citizen2, citizen3, citizen4, citizen5, citizen6 };

        return citizen;
    }

}
```

CitizenFinder.java

- Include the following code formatter and method to capture the data about the citizen to find.

```
    private static DateTimeFormatter INPUT_DATE_FORMAT =
            DateTimeFormatter.ofPattern("dd/MM/yyyy");

    public static Citizen getCitizenToFind() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter National ID");
        int nationalId = scanner.nextInt();
        scanner.nextLine();

        System.out.println("Enter First Name");
        String firstName = scanner.nextLine();

        System.out.println("Enter Last Name");
        String lastName = scanner.nextLine();

        System.out.println("Enter Date of Birth in dd/mm/yyyy format");
        String dateOfBirthText = scanner.nextLine();
        LocalDate dateOfBirth =
                LocalDate.parse(dateOfBirthText, INPUT_DATE_FORMAT);

        return new Citizen(nationalId, firstName, lastName, dateOfBirth);
    }
```

- Include main() method and implement the following steps.
- Invoke the getCitizenToFind() method to get the citizen details that needs to be found.

```
Citizen citizenToFind = getCitizenToFind();
```

- Get the citizen array from CitizenDataManager class:

```
Citizen[] citizens = CitizenDataManager.getCitizenData();
```

- Initialize a Boolean variable to find if a matching citizen was found or not.

```
boolean citizenFound = false;
```

- Implement a for loop that finds the matching citizen:

```
for (Citizen citizen : citizens) {
    if (citizen.getNationalId() == citizenToFind.getNationalId() &&
            citizen.getFirstName().equals(citizenToFind.getFirstName()) &&
            citizen.getLastName().equals(citizenToFind.getLastName()) &&
            citizen.getDateOfBirth().equals(citizenToFind.getDateOfBirth())) {
        System.out.println("Citizen FOUND");
        System.out.println(citizen);
        citizenFound = true;
    }
}
```

- After the for loop implement the logic to display message if a match was not found.

```
if (!citizenFound) {
    System.out.println("Citizen NOT FOUND");
}
```

- Run the program and check the result.

Activity (Understand how objects, assignment and references work)

Understand how relational operator equals work on objects

- Create class com.company.equals.ObjectVerifier
- Implement the following code in main() Method

```
public static void main(String args[]) {
    Citizen citizen1 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));
    Citizen citizen2 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));

    System.out.println(citizen1);
    System.out.println(citizen2);
    String result = (citizen1 == citizen2) ? "EQUAL" : "NOT EQUAL";

    System.out.println(result);
}
```

- Run the program and check the result.

Understand how relational operator equals work on objects

- Open class com.company.equals.ObjectVerifier
- Modify the code in main() Method

```
Citizen citizen1 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));
Citizen citizen2 = new Citizen(888888, "Kate", "Sam", 'F', LocalDate.of(1994, 12, 25));
citizen1 = citizen2;
System.out.println(citizen1);
System.out.println(citizen2);
String result = (citizen1 == citizen2) ? "EQUAL" : "NOT EQUAL";

System.out.println(result);
```

- Run the program and check the result.

Understand how equals() method work

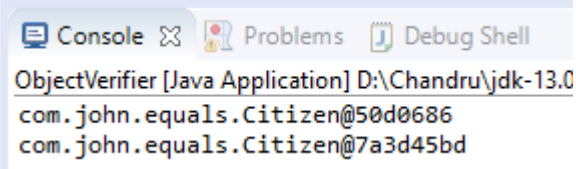
- Comment out the toString() method in Citizen.java.
- Open class com.company.equals.ObjectVerifier
- Modify the code in main() Method

```
Citizen citizen1 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));
Citizen citizen2 = new Citizen(123434, "John", "Sam", 'M', LocalDate.of(1984, 7, 21));

System.out.println(citizen1);
System.out.println(citizen2);
String result = (citizen1.equals(citizen2)) ? "EQUAL" : "NOT EQUAL";

System.out.println(result);
```

- Include a breakpoint in the line where equals() method is called.
- Right click on ObjectVerifier.java and select Debug As… > Java Application
- Accept the perspective switch
- Click "Step Into" option in the toolbar.
- This will navigate to the equals() method in Object class.
- Notice that the equals method in Object class checks for equality of memory reference.

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Notice the citizen details displayed in the console. This result is based on the toString() method implemented in Object class. The memory reference of each object is different.



- Each object instance is a separate copy, hence from JVMs perspective both the objects are different.
- Use the Resume button to continue the program execution.
- Click the below icon in the top right corner to switch back the perspective.

Solution (Using equals())

Citizen.java

- Open class com.company.equals.Citizen
- Uncomment the toString() method.
- Place the cursor before the toString() method.
- Select Source > Generate hashCode() and equals…
- Check "Use blocks in 'if' statements"
- Click "Generate"

CitizenFinder.java

- Include main() method and implement the following steps.

- Modify the if condition inside the for loop as specified below.

```
for (Citizen citizen : citizens) {
    if (citizen.equals(citizenToFind)) {
        System.out.println("Citizen FOUND");
        System.out.println(citizen);
        citizenFound = true;
    }
}
```

- Run the program and check the result.

Activity (Understand how overridden equals() method work)

- Debug the ObjectVerifier code and check which equals() method is called now.

Key takeaways

- Let us recollect that Object is the ultimate parent of any class.
- Object class has implementations for hashCode(), equals() and toString() methods.
- These methods in Object class can be overridden to have class specific implementation.
- The new keyword creates an object and allocates a memory space.
- The assignment operator creates a reference to the created object instance.
- When assigning one object reference to another object reference, only the point changes, but the values remain intact and is not changed.
- Equals relational operator compares the memory reference to check the equality
- Always override equals() method to check equality of objects.
- For checking the matching LocalDate, find that equals() method of LocalDate() is called, which means that for LocalDate class has equals() method implemented. The same is applicable for String class as well.

Questions

- What happens when relational equals operator is used to compare two object instances?
- A class has not implemented equals() method. When equals() method is invoked in this class instance what happens.
- What is the check done in equals() method of Object class?
- Describe the importance and significance of toString() and equals() method in Object class.

Validate input date format for age calculation using Exceptions

Problem

For example, consider entering "33/08/2019" as Date of Birth, this will result in displaying exception refer "Expected Output #1" below. If somebody enters date wrongly, this will not be a good user experience. Display appropriate error message when the date is entered in wrong format. Refer sample output 2 and 3 below.

Expected Output #1 (without exception handling)

```
Enter Date of Birth in dd/mm/yyyy format
33/11/2019
Exception in thread "main" java.time.format.DateTimeParseException: Text '33/11/2019' could not be parsed: Invalid value for DayOfMonth (valid values 1 - 28/31): 33
        at java.base/java.time.format.DateTimeFormatter.createError(DateTimeFormatter.java:2020)
        at java.base/java.time.format.DateTimeFormatter.parse(DateTimeFormatter.java:1955)
        at java.base/java.time.LocalDate.parse(LocalDate.java:428)
        at com.company.date.AgeCalculator.main(AgeCalculator.java:17)
Few more error lines removed.
```

Expected Output #2

```
Enter Date of Birth in dd/mm/yyyy format
11/15/2019
Invalid Date: Text '11/15/2019' could not be parsed: Invalid value for MonthOfYear (valid values 1 - 12): 15
```

Expected Output #3

```
Enter Date of Birth in dd/mm/yyyy format
33/12/2019
Invalid Date: Text '33/12/2019' could not be parsed: Invalid value for DayOfMonth (valid values 1 - 28/31): 33
```

Solution

- Open "com.company.date.AgeCalculator" in Eclipse
- Enclose the last four lines of code in main() method within try/catch block as specified.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
try {
    LocalDate date = LocalDate.parse(input, formatter);

    Period period = Period.between(date, LocalDate.now());
    int age = period.getYears();
    System.out.println("Age: " + age);
} catch (DateTimeParseException e) {
    System.out.println("Invalid Date: " + e.getMessage());
}
System.out.println("After try/catch block");
```

- Compile and run the program.

Explanation

- Exceptions represents an extraordinary situation where in a method during execution is not able to execute an intended operation for various reasons.
- Examples of exceptions in a real-world scenario that affects the normal course of behavior:
  - Experiencing a flat tire when driving a car
  - A power cut when watching your favorite TV program
  - Microphones not working during a stage drama

- Exception handling in Java helps addressing an exception scenario and makes the program take an alternative approach.
- There can be zero or more exceptions associated with a method.
- For example, the parse() method in LocalDate method throws DateTimeParseException. Refer documentation (https://docs.oracle.com/javase/8/docs/api/ > java.time > LocalDate > Click on the second parse() method > Refer the Exception section)
- This exception is thrown when the parsing of the text to LocalDate fails.
- If the exception is not handled the execution of the program abruptly stops and the subsequent lines of the program are not executed. A stack trace is printed on the screen to denote where exactly the execution failed.
- A try/catch block helps in handling the exception.
- When an exception happens in try block, the code within the catch block is executed. The program execution does not stop, but continues execution after execution of the catch block.
- If the exception does not happen, all the lines within the try block is executed, then the lines following the catch block is executed. Kindly note that in case the catch block is not executed.

Exercise Problem

In Training End Date calculation program, there is a possibility that the number of weeks can be entered as character instead of number. Handle the exception that occurs and display an appropriate message to enter number.

Hint: Execute the current program with character as input for number of weeks. Find out the exception that is thrown and try to catch the exception and handle it accordingly.

Reference

Example - https://beginnersbook.com/2013/04/try-catch-in-java/

Exception

- What is the importance of Exception Handling?
- Explain the syntax on how try/catch needs to be implemented?
- What happens when an exception is not handled?
- What is provided in the brackets after the catch keyword?
- What is the purpose of getMessage() method in an exception?

Respond insufficient balance using Custom Exception

Problem

A customer of XYZ Bank tries to withdraw money from the bank's ATM. The respective customer's account balance is 3,000, but tries to withdraw 3,500 unknowingly. The withdraw() method in Account.java is used to perform this transaction, but this method does not have any mechanism to notify that there is insufficient balance in the account. Modify the withdraw() to method to notify the caller of the method that there is insufficient balance, based on which the ATM machine will display appropriate message.

Expected Output #1 (Insufficient Balance)

```
Enter ATM Cash Withdrawal Amount
3500
INSUFFICIENT BALANCE
```

Expected Output #2 (Sufficient balance available)

```
Enter ATM Cash Withdrawal Amount
2000
Balance after Withdrawal is 3000
```

Solution (Using Unchecked Exception)

Account.java

- Open com.xyzbank.model.Account in Eclipse
- Include a new constructor with account number and balance.
- Include getter methods for account number and balance. Follow steps below:
  - Place the cursor after both the constructors
  - Go to Eclipse Menu > Source > Generate Getters and Setters
  - Expand accountNumber and check getAccountNumber()
  - Expand balance and check getBalance()
  - Click Generate
  - This will generate methods to read the values of accountNumber and balance.

SavingsAccount.java

- Include new constructor with account number and balance
- Within this constructor invoke the respective parent constructor using super keyword

TransactionManager.java

- Create new class TransactionManager in com.xyzbank.model package
- Within main() method create instance of SavingsAccount with accountNumber as "003" and balance as 3000. This account represents the customer who is withdrawing from the ATM.

```
SavingsAccount account = new SavingsAccount("003", 3000.0);
```

- Using scanner get the withdrawal amount and invoke the withdraw method.

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter ATM Cash Withdrawal Amount");
double withdrawAmount = scanner.nextDouble();
account.withdraw(withdrawAmount);
```

- Print the balance using getBalance() method:

```
System.out.println("Balance after Withdrawal is " + account.getBalance());
```

- Run TransactionManager class and check the behavior.

InsufficientBalanceException.java

- Create a new class InsufficientBalanceException that extends from RuntimeException class.
- No other implementation is required within this class.

Account.java

- Modify the withdraw() method as specified below:

```
public void withdraw(double transactionAmount) throws InsufficientBalanceException {
    double tentativeBalance = balance - transactionAmount;
    if (tentativeBalance < 0) {
        throw new InsufficientBalanceException();
    }
    balance = tentativeBalance;
}
```

TransactionManager.java

- Modify the main() method code as specified below:

```
public static void main(String[] args) {
    SavingsAccount account = new SavingsAccount("003", 3000.0);

    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter ATM Cash Withdrawal Amount");
    double withdrawAmount = scanner.nextDouble();
    try {
        account.withdraw(withdrawAmount);
        System.out.println("Balance after Withdrawal is " + account.getBalance());
    } catch (InsufficientBalanceException e) {
        System.out.println("INSUFFICIENT BALANCE");
    }
}
```

- Run the program and check if expected results are achived.

Solution using Checked Exception

InsufficientBalanceException.java

- Modify this class to extend from "Exception" instead of "RuntimeException".

```
public class InsufficientBalanceException extends Exception {

}
```

- This will result in compilation errors in Account.java, enclose the withdraw() method within try catch block, with no lines of code within the catch block.
- Run TransactionManager and check if expected results are achieved.

Explanation

- Create an exception by extending Exception class or RuntimeException class.
- Extending Exception class creates a Checked Exception
- Extending RuntimeException class creates an Unchecked Exception
- Define a method with exception using throws keyword in the end of method signature
- Throw an exception using "throw new" keywords followed by the invocation of the exception class constructor.
- Using Checked Exception forces the developer to handle the exception using try/catch block

- Using Unchecked Exception does not force the developer to handle the exception, but if the exception occurs during runtime, the program execution gets halted.

Exercise Problem

Create a class to represent a school student with following attributes:

- ID
- Name
- Standard (Numeric value between 1 to 12)
- Section

Creation of an instance of school student should not create an object if the value for standard is not between 1 and 12.

Reference

https://docs.oracle.com/javase/tutorial/essential/exceptions/

Questions

- How to define an exception?
- What all needs to be done for making a method throw an exception?
- What is Checked Exception?
- What is Unchecked Exception?
- Unchecked Exception forces the developer to handle the exception. (True / False)
- If a code is invoking a method that throws a Checked Exception, what are the list of steps that needs to be implemented.

Solve the Profit Calculation using Wrapper Classes

Problem

Calculate the Profit based on Buying Price and Selling Price using wrapper classes.

Profit = Selling Price – Buying Price

The output should be printed as given below. Hard code value of buyingPrice and sellingPrice as 15 and 25 respectively.

Profit = 10

Solution

- Create a class named ProfitCalculator in package "com.company.wrapper".
- Copy the ProfitCalculator code from the problem that was solve earlier and paste this code into the main() method of the new class.
- Change data type "int" to "Integer"
- Create instances of Integer values using new keyword.

```
Integer buyingPrice = new Integer(15);
Integer sellingPrice = new Integer(25);
Integer profit = sellingPrice - buyingPrice;
System.out.println("Profit = " + profit);
```

- Run the program and check the result.

Solution using valueOf() method

- Modify creation of Integer values using valueOf() method:

```
Integer buyingPrice = Integer.valueOf(15);
Integer sellingPrice = Integer.valueOf(25);
Integer profit = sellingPrice - buyingPrice;
System.out.println("Profit = " + profit);
```

- Run the program and check the result.

Solution using auto-boxing

- Modify creation of Integer values using auto-boxing:

```
Integer buyingPrice = 15;
Integer sellingPrice = 25;
Integer profit = sellingPrice - buyingPrice;
System.out.println("Profit = " + profit);
```

- Run the program and check the result.

Explanation

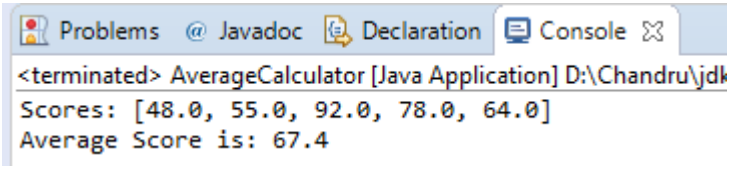
- Use wrapper classes when there is a need to represent primitive data types as objects.
- There is a respective wrapper class defined for each primitive data type.
- As part of JDK 9 release the constructors of wrapper classes had been deprecated. It has recommended to use valueOf() method to improve memory usage and performance.
- Autoboxing was introduced in JDK 5, which does an automatic conversion of primitive data type to wrapper classes.

Compute average score using ArrayList

Problem

For 5 students in classroom, compute the average marks scored by the classroom in science subject. The scores of the students in science subject are 48, 55, 92, 78, 64. The formula to compute the score is to add all the scores and divide it by 5. Solve this problem using ArrayList.

Sample output



Solution

- Create new class com.company.collection.AverageCalculator with main() method.
- Import ArrayList.

```
import java.util.ArrayList;
```

- Include the coding as specified below for creation of ArrayList and adding scores:

```
ArrayList<Double> scores = new ArrayList<>();
scores.add(48.0);
scores.add(55.0);
scores.add(92.0);
scores.add(78.0);
scores.add(64.0);
System.out.println("Scores: " + scores);
```

- Then calculate the average and display the result:

```
double total = 0;
for (double score : scores) {
    total += score;
}
double average = total / scores.size();
System.out.println("Average Score is: " + average);
```

- Run the program and check the result

Explanation

- ArrayList class is part of Collections Framework in java.
- It is available in the java.util package.
- On creation of ArrayList define the type using angular brackets.
- The add() method in ArrayList adds a new item to the end of the ArrayList.
- ArrayList can grow dynamically, whereas the size of an Array is fixed and cannot be changed.
- The double literal passed using the add() method gets converted into java.lang.Double based on the autoboxing feature.
- ArrayList cannot hold primitive data types.
- The size() method in ArrayList class returns the number of items in the ArrayList.
- ArrayList retains the order in which the elements are added.
- ArrayList is extended from AbstractList
- AbstractList is extended from AbstractCollection

- AbstractCollection is extended from Object
- One of the main interface that ArrayList implements is List.

Activity

- Open com.company.collection.AverageCalculator
- Include the below line after System.out.println of the “scores”. This method removes and item.

```
scores.remove(92.0);
System.out.println("Scores: " + scores);
```

- Add the removed item in a different position.

```
scores.add(1, 92.0);
System.out.println("Scores: " + scores);
```

- Display the third item in the list.

```
System.out.println("Third item: " + scores.get(2));
```

- Sort the list in ascending order.

```
System.out.println("Sorting the list...");
Collections.sort(scores);
System.out.println("Scores: " + scores);
```

- Sort the list in descending order.

```
System.out.println("Reverse sorting the list...");
Collections.sort(scores, Collections.reverseOrder());
System.out.println("Scores: " + scores);
```

Find a citizen using ArrayList

Problem

Given the details about a list of citizens, find out a citizen based on National ID, First Name, Last Name and Date of Birth. Following are the details stored about a citizen of a country. Solve this problem using ArrayList.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Riki	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

Expected Output (when citizen match found)

```
Enter National ID
987977
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen FOUND
Citizen [nationalId=987977, firstName=John, lastName=Sam, gender=M, dateOfBirth=2000-10-17]
```

Expected Output (when citizen match not found)

```
Enter National ID
985765
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen NOT FOUND
```

Solution

- Copy Citizen, CitizenDataManager and CitizenFinder class from com.company.equals package and paste it in com.company.collections package.
- Ensure that Citizen class has implemented equals() method without inclusion of comparision for gender.
- Open CitizenDataManager in com.company.collections package
- Change the existing method signature as specified below

```
public static ArrayList<Citizen> getCitizenArrayList() {
```

- Comment out the line that creates the array:

```
//Citizen[] citizen = {citizen1, citizen2, citizen3, citizen4, citizen5, citizen6 };
```

- Create a new array list that will have the type as Citizen and add citizen instances using add method:

```
ArrayList<Citizen> citizens = new ArrayList<>();
citizens.add(citizen1);
citizens.add(citizen2);
citizens.add(citizen3);
citizens.add(citizen4);
citizens.add(citizen5);
citizens.add(citizen6);
```

- The returns the citizens array list.

```
return citizens;
```

- In com.company.collection.CitizenFinder class modify the method invocation to the new method:

```
ArrayList<Citizen> citizens = CitizenDataManager.getCitizenArrayList();
```

- Run CitizenFinder and check if the output contains

Explanation

This exercise primarily demonstrates that any new class that we have implemented can also be part of ArrayList.

Sort Citizen based on First Name

Problem

Given the Citizen data below, display the citizen information sorted by first name.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Jack	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

NOTE: Minor correction made in the data, Riki Elliott changed to Jack Elliott.

Expected Output

```
223463 Jack     Potter   M 1996-12-18
123434 John     Sam      M 1984-07-21
987977 John     Sam      M 2000-10-17
878237 Lily     Charlie  F 2001-04-07
233467 Linda    Johnson  F 1974-11-01
765657 Riki     Elliott  F 1999-09-23
```

Solution

Citizen.java

- Modify the class definition of Citizen as specified below:

```
public class Citizen implements Comparable<Citizen> {
```

- Include the following methods in the end of the Citizen class

```
public void display() {
    System.out.printf("%6d %-8s %-8s %c %s\n", nationalId, firstName, lastName,
            gender, dateOfBirth);
}

public int compareTo(Citizen citizen) {
    return firstName.compareTo(citizen.firstName);
}
```

CitizenFirstNameSorter.java

- Create a new class com.company.collection.CitizenFirstNameSorter and implement main method as specified below

```
public static void main(String args[]) {
    ArrayList<Citizen> citizens = CitizenDataManager.getCitizenArrayList();
    Collections.sort(citizens);

    for (Citizen citizen : citizens) {
        citizen.display();
    }
}
```

- Run the program and check the result.

How is sorting implemented?

- Let us try to sort the number 43, 22, 57
- We have to pick two number
- Find out which is greater, lesser or equal and decide to swap the position or not.
- Copy and paste the following lines of code in compareTo() method of Citizen class.

```
    public int compareTo(Citizen citizen) {
        int result = firstName.compareTo(citizen.firstName);
        System.out.println(this.nationalId + " - " + this.firstName + " "
                + citizen.getNationalId() + " - " + citizen.getFirstName()
                + " (" + result + ")");
        return result;
    }
```

- Execute the program and check the result. The result should look something like the below:

```
223463 - Jack 123434 - John (-14)
878237 - Lily 223463 - Jack (2)
878237 - Lily 123434 - John (2)
233467 - Linda 123434 - John (2)
233467 - Linda 878237 - Lily (2)
765657 - Jack 878237 - Lily (-2)
765657 - Jack 123434 - John (-14)
765657 - Jack 223463 - Jack (0)
987977 - John 123434 - John (0)
987977 - John 233467 - Linda (-2)
987977 - John 878237 - Lily (-2)
```

- The Collections.sort() method compares different Citizen class instances by calling the compareTo() method in Citizen.
- The compareTo() method of String class returns a +ve, -ve or zero based on the whichever should come before the other.

Explanation

- The sort method in collection expects an object that is Comparable, hence we have implemented Comparable.
- The Comparable interface enforces to implement the compareTo() method.
- This method is used an invoked within the sort() method to compare two Citizen objects.
- The compareTo() method is supposed to return -1, 0 or 1.
- The value -1 represent that the compared object is smaller
- The value 0 represents that both the objects are equal
- The value 1 represents that the compared object is greater
- In this example the compareTo() method of String object is used, which returns either -1, 0 or 1. Find this the value returned in debug mode.

Questions

- What is the use of Comparable interface?
- What is the expected implementation in compareTo() method and what does it return?
- From where is the compareTo() method called?
- What happens to the collection object after invocation of sort() method?
- Where is the sorting algorithm implemented?

Sort Citizen based on First Name and Last Name

Problem

Given the Citizen data below, display the citizen information sorted by first name and last name. If the first name is same for two citizens then the sorting needs to be done in last name.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Jack	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

Expected Output

```
765657 Jack     Elliott  F 1999-09-23
223463 Jack     Potter   M 1996-12-18
123434 John     Sam      M 1984-07-21
987977 John     Sam      M 2000-10-17
878237 Lily     Charlie  F 2001-04-07
233467 Linda    Johnson  F 1974-11-01
```

Solution

CitizenNameComparator.java

- Create a new class company.collection.CitizenNameComparator

```
public class CitizenNameComparator implements Comparator<Citizen> {
```

- Include the following code in the comparator class:

```
public int compare(Citizen citizen1, Citizen citizen2) {
    if (citizen1.getFirstName().equals(citizen2.getFirstName())) {
        return citizen1.getLastName().compareTo(citizen2.getLastName());
    } else {
        return citizen1.getFirstName().compareTo(citizen2.getFirstName());
    }
}
```

CitizenNameSorter.java

- Create a new class com.company.collection.CitizenNameSorter and implement main method as specified below.

```
public static void main(String args[]) {
    ArrayList<Citizen> citizens = CitizenDataManager.getCitizenArrayList();
    Collections.sort(citizens, new CitizenNameComparator());

    for (Citizen citizen : citizens) {
        citizen.display();
    }
}
```

- Execute this class and check if it gives the expected result.

Explanation

About compare() method logic:

- In the compare() method, first we are checking if the firstName values are not same.
- If firstName values are not same, then sorting needs to be done based on firstName.
- If firstName values are same, then sorting needs to be done based on lastName.

About Comparator

- Use Comparator interface when there is a need to sort objects on difference instance variable at various points of time dynamically.
- Implement Comparator interface with a class having a compare() method with two object references as parameters. Return a +ve, -ve or 0 value based on the object comparison.
- There can be any number of comparators defined for a class.
- The Comparator can be used to compare a single instance variable or multiple instance variables.
- The respective Comparator instance is given as parameter to the sort() method based on whatever sorting is required at that point in this.

Questions

- When do we use a Comparator?
- What are the parameters of compare() method in Comparator implementation class?
- What is the return type of compare() method?
- What is the logic that needs to be implemented within the compare() method?
- What does the compare() method return?

Find a citizen using HashSet

Problem

Given the details about a list of citizens, find out if a citizen exists based on National ID, First Name, Last Name and Date of Birth. Following are the details stored about a citizen of a country. Solve this problem using HashSet.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Riki	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

Expected Output (when citizen match found)

```
Enter National ID
987977
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen FOUND
```

Expected Output (when citizen match not found)

```
Enter National ID
985765
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Citizen NOT FOUND
```

Solution

- Create a new method in CitizenDataManager with the following method signature.

```
public static HashSet<Citizen> getCitizenHashSet() {
```

- Copy and paste the citizen creation code from the previous method.
- Create a new HashSet as specified below:

```
HashSet<Citizen> citizens = new HashSet<>();
```

- Let the add lines remain the same.
- Create a new class CitizenHashSetFinder
- Copy the class code from CitizenFinder and paste it.

- Modify the main method with following code:

```
public static void main(String args[]) {
    HashSet<Citizen> citizens = CitizenDataManager.getCitizenHashSet();
    Citizen citizenToFind = getCitizenToFind();
    if (citizens.contains(citizenToFind)) {
        System.out.println("Citizen FOUND");
    } else {
        System.out.println("Citizen NOT FOUND");
    }
}
```

- Execute the main method and check if appropriate message is displayed.

Explanation

- HashSet can contain only unique items.
  - Add one more citizen in CitizenDataManager.getCitizenHashSet()

```
Citizen citizen7 = new Citizen(987977, "John", "Sam", 'M', LocalDate.of(2000, 10, 17));
```

  - Add this item into the citizens HashSet, then display size and items.

```
citizens.add(citizen7);

System.out.println("Size of HashSet: " + citizens.size());
for (Citizen citizen : citizens) {
    citizen.display();
}
```

  - This should display only 6 items, since citizen1 and citizen7 are equal, citizen7 is not added.
- HashSet is an unordered collection. It does not maintain the order in which the items are inserted. Refer the output of previous activity.
- HashSet uses HashMap to internally store the data. HashMap will be covered as part of next exercise.
- HashSet is not thread safe.
- Include the following line in the hashCode() method of Citizen class. Include this line just before return statement:

```
System.out.println("Hashcode for '" + nationalId + "' is " + result);
```

- Include the following line as the first line in equals() method of Citizen:

```
System.out.println("Inside Equals.");
```

- Include this log after all Citizen objects are created in CitizenDataManager.getCitizenHashSet()

```
System.out.println("Citizen objects created.");
```

- Include this log after adding a citizen to HashSet in CitizenDataManager.getCitizenHashSet()

```
System.out.println("After adding citizen1.");
```

- Execute CitizenHashSetFinder.java. Following is the expected output.

```
Citizen objects created.
Hashcode for '123434' is -1264446910
After adding citizen1.
Hashcode for '223463' is 735961097
Hashcode for '878237' is 1317560335
Hashcode for '233467' is 677418146
Hashcode for '765657' is -107084265
Hashcode for '987977' is -281790171
```

```
Hashcode for '987977' is -281790171
Inside Equals.
Size of HashSet: 6
123434 John     Sam      M 1984-07-21
987977 John     Sam      M 2000-10-17
233467 Linda    Johnson  F 1974-11-01
223463 Jack     Potter   M 1996-12-18
878237 Lily     Charlie  F 2001-04-07
765657 Jack     Elliott  F 1999-09-23
Enter National ID
987977
Enter First Name
John
Enter Last Name
Sam
Enter Date of Birth in dd/mm/yyyy format
17/10/2000
Hashcode for '987977' is -281790171
Inside Equals.
Citizen FOUND
```

- Inference from the output:
  - HashSet internally invokes hashCode() method of Citizen object when adding an item.
  - HashSet stores this hashCode() internally creates a mapping between the hash code and the object
  - When contains method is called, it generates the hash code for the passed object and quickly picks the object based on the hash code, then it runs the equals method ensure that the values are same.
  - HashSet does not do a sequential check on each item and does not call the equals method in each object.

Find a citizen based on National ID using HashMap

Problem

Based on National ID get the details about a Citizen.

Citizen Data

National ID	First Name	Last Name	Gender	Date of Birth
123434	John	Sam	M	21/07/1984
223463	Jack	Potter	M	18/12/1996
878237	Lily	Charlie	F	07/04/2001
233467	Linda	Johnson	F	01/11/1974
765657	Riki	Elliott	F	23/09/1999
987977	John	Sam	M	17/10/2000

Expected Output (when citizen match found)

```
Enter National ID
987977
Citizen FOUND
Citizen [nationalId=987977, firstName=John, lastName=Sam, gender=M, dateOfBirth=2000-10-17]
```

Expected Output (when citizen match not found)

```
Enter National ID
985765
Citizen NOT FOUND
```

Solution

CitizenDataManager.java

- Create a new method with the following signature:

```
public static HashMap<Integer, Citizen> getCitizenHashMap() {
```

- Copy and paste the citizen creation code from the previous method.
- Create a new HashMap and add citizens. Refer code below:

```
HashMap<Integer, Citizen> citizens = new HashMap<>();
citizens.put(citizen1.getNationalId(), citizen1);
citizens.put(citizen2.getNationalId(), citizen2);
citizens.put(citizen3.getNationalId(), citizen3);
citizens.put(citizen4.getNationalId(), citizen4);
citizens.put(citizen5.getNationalId(), citizen5);
citizens.put(citizen6.getNationalId(), citizen6);
```

- Return the citizens HashMap

CitizenHashMapFinder.java

- Create a new class com.company.collection.CitizenHashMapFinder with main method.
- Within main method get nationalId using Scanner.

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter National ID");
int nationalId = scanner.nextInt();
scanner.nextLine();
```

- Get the citizen HashMap:

```
HashMap<Integer, Citizen> citizens = CitizenDataManager.getCitizenHashMap();
```

- Get the Citizen object reference using get() method and display citizen details.

```
Citizen citizen = citizens.get(nationalId);
if (citizen != null) {
    System.out.println("Citizen FOUND");
    System.out.println(citizen);
} else {
    System.out.println("Citizen NOT FOUND");
}
```

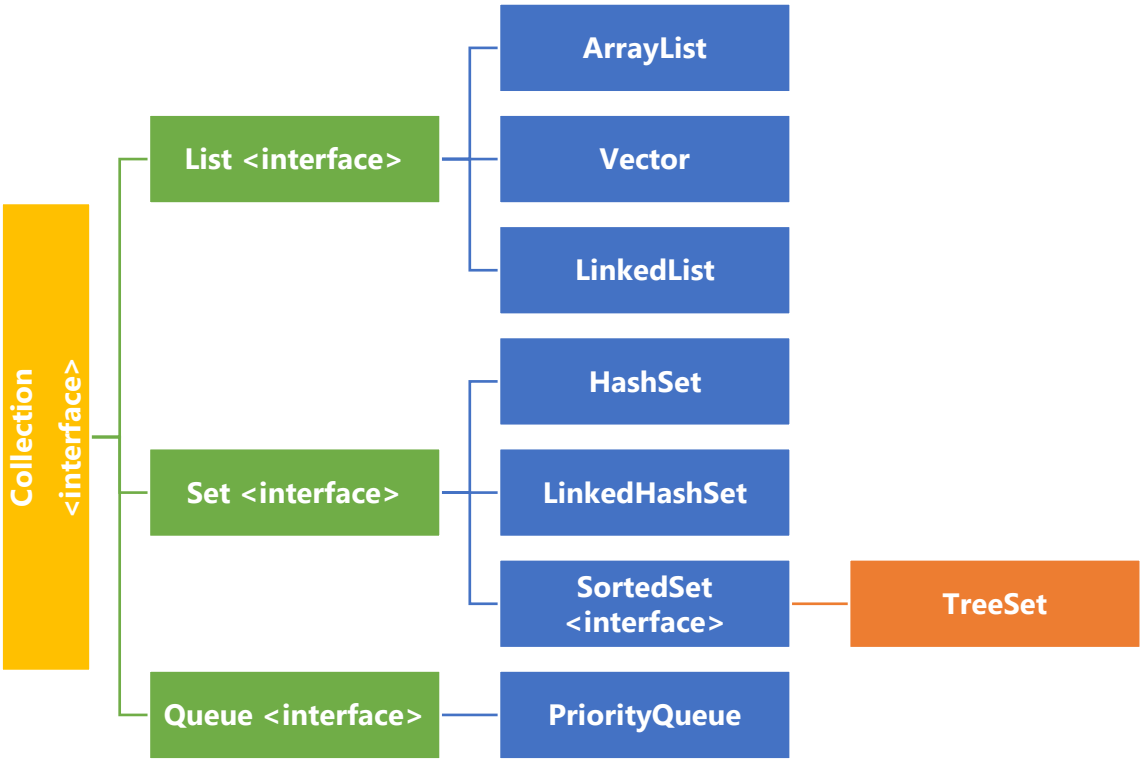
- Execute the program and check if it gets the expected result.

Explanation

- HashMap is a map based collection for storing key & value pairs.
- HashMap does not retain the order of the items.
- HashMap is unsynchronized
- HashMap allows null value as key.
- containsKey() method determines if key is present based on hashCode() and equals() method.
- containsValue() method determines if a value is present based on hashCode() and equals() method.
- get() method return he value based on the key passed as input
- isEmpty() method checks if the HashMap has any entries or not
- keySet() returns the keys as a Set
- put() method adds an item into HashMap
- size() returns the number of key-value mappings available in a HashMap
- remove() method removes an item from the list

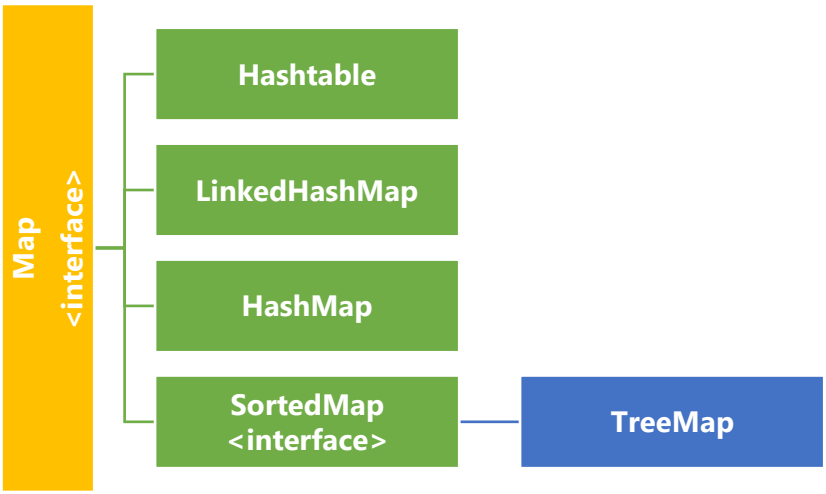
Collections Framework

Collections



- ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet and PriorityQueue are not synchronized, which means it is not thread safe. Multiple threads can modify the content of the collection.
- Vector is synchronized, which means it is thread safe. Only one thread can modify the content. This might reduce the performance, since it might lead to a situation where multiple threads are waiting completion of a thread.

Map



- HashMap, LinkedHashMap and TreeMap are not synchronized and not thread safe.
- Hashtable is thread safe.

Create copy of text file

Problem

Create a text file name "file.txt" with the below content and save it in d drive. Create a copy of this file in the same folder with the name as "file-copy.txt".

file.txt

```
File to test copy.
```

Solution

- Create a new package com.company.io
- Create a file named file.txt in d drive with the text content "`File to test copy.`".
- Create a new class TextFileCopier and include the following code in main method:

```
public static void main(String args[])
        throws FileNotFoundException, IOException {
    FileReader reader = new FileReader("D:\\file.txt");
    FileWriter writer = new FileWriter("D:\\file-copy.txt");

    int character = reader.read();
    while (character != -1) {
        System.out.print(character + " ");
        writer.write(character);
        character = reader.read();
    }

    reader.close();
    writer.close();
}
```

- Execute the class and check if file is copied. Verify the content of the generated file.

Explanation

- FileReader class helps in reading text file.
- This class is part of java.io package.
- The FileReader class can read a single character or as an array of bytes.
- The read() method in FileReader helps to read the byte value of a single character of the file. The byte value provided as primitive int type.
- After executing the read() once, if we call read() again it returns the next character's integer value.
- The read() method returns -1 once the end of file is reached.
- The FileNotFoundException is thrown when the file is not available in the specified folder.
- The IOException will be thrown when there an error when reading the data. Since this file is an external resource available outside the JVM, there are good possibilities that it might get altered or moved or deleted. To handle this kind of a scenario we have IOException.
- FileWriter class helps in writing content to a text file.
- The method close() ensures that the files are closed, so that it is available for other resources.

Create copy of an image file

Problem

A google logo image file needs to be copied into a different name in the same folder. The google logo images available in the link below:

https://upload.wikimedia.org/wikipedia/commons/thumb/5/53/Google_%22G%22_Logo.svg/235px-Google_%22G%22_Logo.svg.png

Source File: D:\google.png

Destination File: D:\google-copy.png

Solution

- Open the below URL in browser

  https://upload.wikimedia.org/wikipedia/commons/thumb/5/53/Google_%22G%22_Logo.svg/235px-Google_%22G%22_Logo.svg.png
- Right click the image and select "Save image as"
- Save the file in d drive with name "google.png"
- Create a new class com.company.io.ImageFileCopier.
- Copy the main method code from TextFileCopier and paste the code here.
- Change the file name references to point to google.png and google-copy.png
- Execute the program
- This will create a copy of the file.
- Open the generated file google-copy.png browser
- The image will not get rendered, since we used FileReader and FileWriter which is meant for dealing with characters, the integrity of the file is logs.
- Change FileReader references to FileInputStream.
- Change FileWriter references to FileOuputStream.
- Modify the file names as "google.png" and "google-copy.png"
- Execute and check if the file copy works.

List all java source and class files implemented before using Eclipse

Problem

Before using Eclipse, we placed all the source files in a specific folder. From this folder list all the java source file names and class file names.

Solution

- Create a new class com.company.io.FileLister.
- Implement the following code:

```
public static void main(String args[]) {
    File file = new File("D:\\sessions\\java-learning");
    String[] list = file.list();

    for (String item : list) {
        System.out.println(item);
    }
}
```

- Modify the folder in the first line with the folder name in your desktop.
- Run the program and check if the file names are listed.

Explanation

- File class helps in navigating the file and folder structure in the system.
- An instance of File class represents a file or a folder in the file system.
- The following are few operations that can be performed with file class:
    - createNewFile() method creates a new file.
    - delete() method deletes the file.
    - exists() method checks if a folder or file exists or not.
    - getPath() returns the path of the file or folder.
    - listFiles() returns all the files in a folder as a File[] array.
    - mkdir() creates a new folder

Generate Credit Card Bill files using Reader classes

Problem

Generate monthly bill for each credit card customer. The credit card transactions done by each customer is available in a data file. Refer content of a sample data file below. This file contains transactions from various customers. The transaction entries need to be grouped according to a specific customer and bill needs to be generated. Refer sample generated bill for a specific customer.

Card Transaction Data File

```
5454252571710001,10/07/2019,5000
5454252571710002,11/07/2019,4000
5454252571710003,12/07/2019,3000
5454252571710001,13/07/2019,2000
5454252571710002,14/07/2019,1000
5454252571710003,15/07/2019,6000
5454252571710001,16/07/2019,7000
5454252571710002,17/07/2019,8000
5454252571710003,18/07/2019,1000
5454252571710001,19/07/2019,4000
5454252571710002,20/07/2019,3000
5454252571710003,21/07/2019,2000
5454252571710001,22/07/2019,1000
5454252571710002,22/07/2019,6000
5454252571710003,23/07/2019,7000
5454252571710001,23/07/2019,8000
5454252571710002,23/07/2019,1000
5454252571710003,24/07/2019,4000
5454252571710001,25/07/2019,3000
5454252571710002,27/07/2019,2000
```

Expected Output (sample bill generated for one card)

```
Credit Card Statement for 5454252571710001
Month:  Jul 2019
------------------
Date        Amount
------------------
10/07/2019 5000.00
13/07/2019 2000.00
16/07/2019 7000.00
19/07/2019 4000.00
22/07/2019 1000.00
23/07/2019 8000.00
25/07/2019 3000.00
------------------
Due       30000.00
------------------

Last Payment Date is 15 July 2019
```

Solution (Step #1: read data file)

Transaction.java

- Create package com.company.card.model
- Create class Transaction

- Implement constructor.

```
public Transaction(String transaction) {
    String fields[] = transaction.split(",");
    this.cardNumber = fields[0];
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    this.transactionDate = LocalDate.parse(fields[1], formatter);
    this.transactionAmount = Double.valueOf(fields[2]);
}
```

- Generate getters setters
- Implement the method below. This will be used to have each transaction printed.

```
public String getCardBillEntry() {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    String transactionDate = this.transactionDate.format(formatter);
    return String.format("%s %6d", transactionDate, transactionAmount);
}
```

TransactionReader.java

- Create TransactionReader class in package com.company.card.
- Implement method below to get the data from file.

```
public static void getData() throws FileNotFoundException, IOException {
    FileReader reader = new FileReader("D:\\transaction.dat");
    int character = reader.read();
    while (character != -1) {
        System.out.println(character);
        character = reader.read();
    }
}
```

BillGenerator.java

- Create BillGenerator class in package com.company.card.
- Implement main method to invoke the method to get data.

```
public static void main(String args[])
        throws FileNotFoundException, IOException {
    TransactionReader.getData();
}
```

- Execute BillGenerator.
- This will result in error since transaction.dat file is not available in the folder.

```
public static void main(String args[])
        throws FileNotFoundException, IOException {
    TransactionReader.getData();
}
```

- Modify getData() method in TransactionReader to handle exceptional scenarios.

```
public static void getData() {
    FileReader reader;
    String fileName = "D:\\transaction.dat";
    try {
        reader = new FileReader(fileName);
        int character = reader.read();
        while (character != -1) {
            System.out.println(character);
            character = reader.read();
        }
    } catch (FileNotFoundException e) {
        System.out.println("File '" + fileName + "' not found.");
    } catch (IOException e) {
        System.out.println("Error when reading file '" + fileName + "'.");
    }
}
```

Creating the data file

- Create BillGenerator class in package com.company.card.
- Copy the below transaction date from the problem statement and paste it in the new Notepad++ file
- Create a new folder in D: drive with folder name "data".
- Save the file in D: with name 'transaction.dat' in the "data" folder.
- Now run BillGenerator class, this will display numbers in each line.

Explanation

- For each entry in the data file one Transaction instance needs to be created. Each line in the data file will be passed to Transaction constructor to create an instance.
- The getCardBillEntry() method in Transaction will be used to display each transaction in the bill.
- FileReader class helps in reading text file.
- This class is part of java.io package.
- The FileReader class can read a single character or as an array of bytes.
- The read() method in FileReader helps to read the byte value of a single character of the file. The byte value provided as primitive int type.
- After executing the read() once, if we call read() again it returns the next character's integer value.
- The read() method returns -1 once the end of file is reached.
- The FileNotFoundException is thrown when the file is not available in the specified folder.
- The IOException will be thrown when there an error when reading the data. Since this file is an external resource available outside the JVM, there are good possibilities that it might get altered or moved or deleted. To handle this kind of a scenario we have IOException.
- The two exceptions are handled and appropriate error message is displayed, so that whom ever is going to use this program will know the reason for failure.
- The number displayed is the numeric ASCII value for each character. Cast it to char type to display the character.

```
System.out.println((char) character);
```

- Casting is the process to convert a data type to another data type.

Solution (Step 2: read data file and display the content line by line)

TransactionReader.java

- Initialize BufferedReader in the first line of getData() method;

```
BufferedReader reader;
```

- Modify the line that creates a FileReader to wrap with BufferedReader.

```
reader = new BufferedReader(new FileReader(fileName));
```

- Instead of read() method use readLine() and assign it to a String variable and modify the while loop to use readLIne() and print the read line.

```
String line = reader.readLine();
while (line != null) {
    System.out.println(line);
    line = reader.readLine();
}
```

- Execute BillGenerator and check if the lines from the file is printed.
- Check if the lines from the file are printed.

Explanation

- BufferedReader class is from java.io package
- This class contains methods to read a line, hence we are wrapper the FileReader with BufferedReader to read line by line.

Solution (Step 3: populate HashMap based on the data read from file)

TransactionReader.java

- Include a new HashMap static variable to store the card and transaction data read from the file.

```
private static HashMap<String, ArrayList<Transaction>> cardTransactions =
        new HashMap<>();
```

- Create a new method to populate the HashMap:

```
private static void addTransactionToMap(Transaction transaction) {
    String cardNumber = transaction.getCardNumber();
    if (cardTransactions.containsKey(cardNumber)) {
        ArrayList<Transaction> transactions = cardTransactions.get(cardNumber);
        transactions.add(transaction);
    } else {
        ArrayList<Transaction> transactions = new ArrayList<>();
        transactions.add(transaction);
        cardTransactions.put(cardNumber, transactions);
    }
}
```

- Modify the while loop to use the above method to populate the HashMap.

```
while (line != null) {
    Transaction transaction = new Transaction(line);
    addTransactionToMap(transaction);
    line = reader.readLine();
}
```

Explanation

- For each entry in the data file one Transaction instance needs to be created. Each line in the data file will be passed to Transaction constructor to create an instance.
- Each instance of Transaction must be grouped under each Card Number. For managing this data we are using a HashMap with key as Card Number and value has an ArrayList of Transactions.

```
private static HashMap<String, ArrayList<Transaction>> cardTransactions =
        new HashMap<>();
```

- Whenever a new card number is encountered a new ArrayList<Transaction> is created and the transaction instance is added to this array list.

```
        } else {
            ArrayList<Transaction> transactions = new ArrayList<>();
            transactions.add(transaction);
            cardTransactions.put(cardNumber, transactions);
        }
    }
```

- If card number already exists in HashMap then a new entry is added to the array list of that card number.

```
        if (cardTransactions.containsKey(cardNumber)) {
            ArrayList<Transaction> transactions = cardTransactions.get(cardNumber);
            transactions.add(transaction);
        } else {
```

Solution (Step 4: Generate bill for each Card)

CardBillWriter.java

- Create new class CardBillWriter in com.company.card package
- Extend this class from BufferedWriter

```
public class CardBillWriter extends BufferedWriter {
```

- Include the following instance variables

```
private FileWriter fileWriter;
private String cardNumber;
private ArrayList<Transaction> transactions;
private double total;
```

- Include the following method to print headers

```
private void writeHeader() throws IOException {
    super.write("Credit Card Statement for " + cardNumber);
    super.newLine();
    super.write("Month:  Jul 2019");
    super.newLine();
    super.write("------------------");
    super.newLine();
    super.write("Date        Amount");
    super.newLine();
    super.write("------------------");
    super.newLine();
}
```

- Include the following method to print transactions

```
private void writeTransactions() throws IOException {
    for (Transaction transaction : transactions) {
        super.write(transaction.getCardBillEntry());
        super.newLine();
        total += transaction.getTransactionAmount();
    }
}
```

- Include the following method to print total

```
private void writeTotal() throws IOException {
    super.write("------------------");
    super.newLine();
    super.write(String.format("Due       %7.2f", total));
    super.newLine();
    super.write("------------------");
    super.newLine();
}
```

- Include the following method to print footer

```
private void writeFooter() throws IOException {
    super.newLine();
    super.write("Last Payment Date is 15 July 2019");
    super.newLine();
}
```

- Include the following method to print the entire bill

```
public void generateBill() {
    try {
        writeHeader();
        writeTransactions();
        writeTotal();
        writeFooter();
        super.close();
    } catch (IOException e) {
        System.out.println("Error when writing file for " + cardNumber);
    }
}
```

BillGenerator.java

- Include new method to generate bill for all cards.

```
public void generateAllBills() {
    HashMap<String, ArrayList<Transaction>> cardTransactions =
            TransactionReader.getData();

    for (String cardNumber : cardTransactions.keySet()) {
        ArrayList<Transaction> transactions = cardTransactions.get(cardNumber);
        try {
            CardBillWriter writer = new CardBillWriter(cardNumber, transactions);
            writer.generateBill();
            System.out.println("Bill generated for " + cardNumber);
        } catch (IOException e) {
            System.out.println("Error creating file for " + cardNumber);
        }
    }
}
```

- Modify the main method to instantiate BillGenerator and generate bills.

```
public static void main(String args[])
        throws FileNotFoundException, IOException {
    BillGenerator generator = new BillGenerator();
    generator.generateAllBills();
}
```

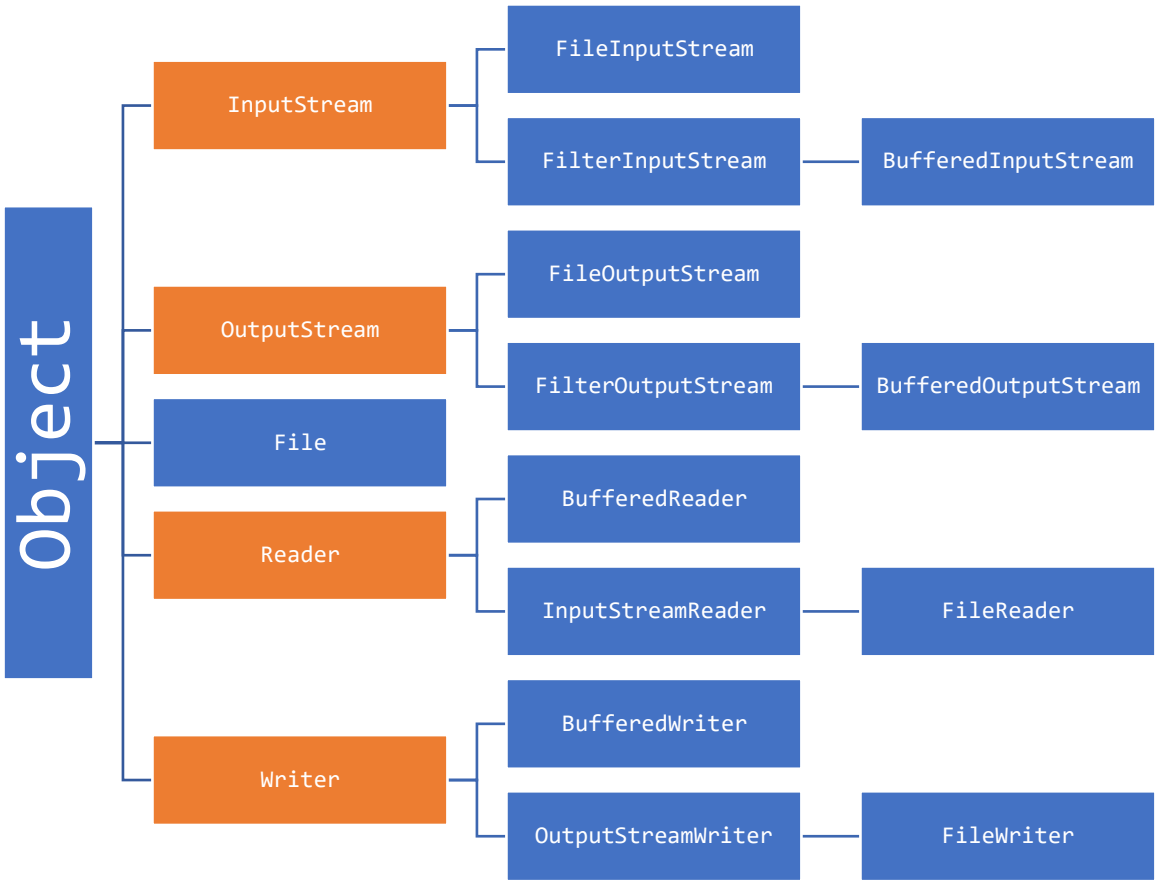
Explanation

- In this part of coding we have reused file writing by extending BufferedWriter.
- The newline() method helps in OS specific new line implementation, hence we are not directly writing the new line.
- The write() method writes the text content into the file.
- The IOException happens if write() method fails for any possible reasons (non-availability of disk space, insufficient privileges to write files)
- The close() method closes and releases the file.

Java Input / Output

Overview

Find below the high-level view about a subset of classes in java.io package.



Items marked in orange represent Abstract Classes

Explanation

- Input and Output stream is used to reading files as bytes.
- Reader and Writer classes are used to read files with characters.
- The java.nio package is a different package for handling streams using alternate methods.

Optimize bill processing for a telecom company using threads

Problem

The post-paid bill processing of a telecom company is having performance issues. The post-paid bill generation for 10,000 customers takes approximately 3 hours. It takes 1 second to generate post-paid bill for one customer. The overall processing time needs to be reduced.

Solution (implementation without thread)

- Create a new package com.company.thread.
- Create a new class BillProcessor and implement the code below.

```
public class BillProcessor {

    private long mobileNumber;

    public BillProcessor(long mobileNumber) {
        this.mobileNumber = mobileNumber;
    }

    public void process() {
        try {
            System.out.println("Start " + mobileNumber);
            Thread.sleep(1000);
            System.out.println("End   " + mobileNumber);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

- Create a new class BillGenerator and implement the code below.

```
import java.time.Duration;
import java.time.LocalDateTime;

public class BillGenerator {

    public static void main(String args[]) {

        LocalDateTime start = LocalDateTime.now();
        for (int i = 0; i < 5; i++) {
            BillProcessor processor = new BillProcessor(i);
            processor.process();
        }
        LocalDateTime end = LocalDateTime.now();
        System.out.println("Duration in seconds: "
                + Duration.between(start, end).getSeconds());
    }

}
```

- Run the code and check the duration of execution.

Explanation

- BillProcessor class is used for processing a bill for one mobile number.
- The sleep() call is used to simulate the 1 minute time taken to process one bill.
- The value 1000 passed to the method represents the time in milliseconds. One second has 1000 milliseconds.

- For the sake of simplicity, the mobile number is generated as numbers starting from 1.
- In the for loop five numbers are created.
- For each mobile number a new BillProcessor instance is created and process() method is called.
- Hence, the overall program execution takes 5 seconds to complete.
- The program completed so far demonstrates the current scenario.

Solution (convert into a thread-based implementation)

- Extend BillProcessor from Thread
- Include a new method run() in the BillProcessor class.

```
public void run() {
    process();
    System.out.println("run() method called");
}
```

- In BillGenerator class call start() method instead of process() method.

```
processor.start();
```

- Temporarily comment the for loop to execute only once.

```
//for (int i = 0; i < 1; i++) {
    BillProcessor processor = new BillProcessor(1);
    processor.start();
//}
```

- Run BillGenerator and check the result, which should look something like the below.

```
Start 0
Duration in seconds: 0
End   0
run() method called
```

- The result obtain may vary based on each system. Even executing the same program again might not get the same result.

Explanation

- BillProcessor class is converted into a thread by extending Thread class.
- Thread class belongs to java.lang package.
- The run() method of Thread had been overridden.
- Notice that we have not invoked the run(), but it is still called. The invocation of start() method available in parent Thread class executes the overridden run() method.
- When start() method is called, a new thread execution is initiated in parallel.
- At this point of time, there will 2 threads in execution:
    - One is the BillGenerator class main method. The lines after the start() method call gets executed in parallel.
    - Another is the BillProcessor class run() method which in turn calls process method.
- As there is time delay given in process method, it takes time to complete, in the mean time the main() method execution completes, we can find that by seeing the duration printed before the end of process() method.
- If we want the main() method to wait till the other thread completes, then we can use the join() method of the thread.

- Include the join() method call before getting the end time.

```
try {
    processor.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
LocalDateTime end = LocalDateTime.now();
```

- Now convert the program to execute five time and check the result. Make below specified modifications.
- Include throws in main() method:

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

- Create an array list to hold all the threads before the for loop.

```
ArrayList<BillProcessor> processors = new ArrayList<>();
for (int i = 0; i < 5; i++) {
    BillProcessor processor = new BillProcessor(i);
    processor.start();
    processors.add(processor);
}

for (BillProcessor processor : processors) {
    processor.join();
}
```

- Remove sys out in run() method.
- Execute BillGenerator. The output should look something like the below.

```
Start 1
Start 2
Start 3
Start 4
Start 0
End   1
End   2
End   0
End   4
End   3
Duration in seconds: 1
```

- The output will vary each time it is executed. But find that the overall duration has taken only 1 second, earlier it was taking 5 seconds to process the same number of bills.
- Modify the number of iterations from 5 to 10,000. Run the program and check the overall time taken to process all the records, which is significantly lesser than the current processing duration of 3 hours.

Solution (implement using Runnable interface)

- Create new class com.company.thread.BillProcessorRunnable

  https://raw.githubusercontent.com/jjchandru/java-learn/master/src/com/company/thread/BillProcessorRunnable.java
- Copy and paste the class content from BillProcessor.
- Modify the constructor name to BillProcessorRunnable
- Create new class com.company.thread.BillGeneratorRunnable.

  https://raw.githubusercontent.com/jjchandru/java-learn/master/src/com/company/thread/BillGeneratorRunnable.java
- Modify the code in main method of BillGeneratorRunnable as specified below.

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

    LocalDateTime start = LocalDateTime.now();
    ArrayList<Thread> threads = new ArrayList<>();
    for (int i = 0; i < 10000; i++) {
        BillProcessorRunnable processor = new BillProcessorRunnable(i);
        Thread thread = new Thread(processor);
        thread.start();
        threads.add(thread);
    }

    for (Thread thread : threads) {
        thread.join();
    }
    LocalDateTime end = LocalDateTime.now();
    System.out.println("Duration in seconds: "
            + Duration.between(start, end).getSeconds());
}
```

- Execute the class and check the result.

Explanation

- Instead of extending from thread class, here we are implementing Runnable interface.
- In BillGeneratorRunnable class use Thread class to create a new instance of threads.
- Include the threads in arraylist and join the same instead of BillProcessor.
- Follow steps below to increase the volume of operations.
    - Open Task Manager
    - Click Performance tab and check the number of Threads currently running.
    - In BillGeneratorRunnable class change the value 10,000 to 30,000 and run.
    - Go to Task Manager and check that the thread counts increase drastically.
    - There are possibilities that the operating system might deny providing threads if its thread limit exceeds, which might result in a fatal error.
    - Use thread pooling option available in Java API to avoid misusing operating system resources. Refer example in the next section.
    - Revert back the count to threads to 10,000.

Solution (implement using Thread executor service)

- Create new class com.company.thread.BillGeneratorThreadPool

  https://raw.githubusercontent.com/jjchandru/java-learn/master/src/com/company/thread/BillGeneratorThreadPool.java
- Modify the code in main method of BillGeneratorThreadPool as specified below. Have this code in between the start and end time definition.

```
ExecutorService executor = Executors.newFixedThreadPool(1000);
for (int i = 0; i < 10000; i++) {
    Runnable worker = new BillProcessorRunnable(i);
    executor.execute(worker);
}
executor.shutdown();
while (!executor.isTerminated()) {
}
```

- When executing this class, not more than 1000 threads would be used.

Understanding Synchronized

- Create new class com.company.thread.Plan

  https://raw.githubusercontent.com/jjchandru/java-learn/master/src/com/company/thread/Plan.java

- Create a method as specified below:

```
public synchronized static Plan getPlan() throws InterruptedException {
    Thread.sleep(1000);
    return null;
}
```

- Invoke this method in BillProcessorRunnable in process method in place of Thread.sleep().

```
public void process() {
    try {
        System.out.println("Start " + mobileNumber);
        Plan.getPlan();
        System.out.println("End   " + mobileNumber);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- Modify the number of threads in BillGeneratorRunnable as 10.
- Run and check the result, it should take 10 minutes to complete.

Explanation

- The 'synchronized' keyword helps in ensuring that only one thread runs at any given point of time.
- The 'synchronized' keyword can be applied in the following areas:
  - Instance methods
  - Static methods
  - Code block inside instance method
  - Code block inside static method
- For instance and static methods, it is sufficient to include the 'synchronized' keyword in the method signature, like how to defined it in the getPlan() method.
- Refer example below for defining 'synchronized' keyword in code blocks of either instance methods or static methods.

```
synchronized(this) {

}
```

- Let us understand how Vector class is defined as synchronized.
- In Eclipse
  - Expand the java-learn project
  - Expand JRE System Library
  - Expand java.base
  - Expand java.util
  - Expand Vector
  - Double click on add(E) method, one can find the method is defined as 'synchronized'.

```
public synchronized boolean add(E e) {
    modCount++;
    add(e, elementData, elementCount);
    return true;
}
```

- Similarly check the add() method in ArrayList, which will not contain the 'synchronized' keyword.

```
public boolean add(E e) {
    modCount++;
    add(e, elementData, size);
    return true;
}
```

Examples of Threads

- A telephonic conversation is synchronous
- Posting a message in social media is asynchronous.
- Sending a message in WhatsApp is asynchronous.
- Opening multiple tabs in a browser with each tab opening a different web site.
- Running multiple apps simultaneously in a mobile phone.

Check your understanding

- What are threads?
- What are the benefits of using a thread?
- What are the two way in which threading can be implemented in Java?
- List the steps to initiate a thread when using the Thread class extension option.
- List the steps to initiate a thread when implementing runnable.
- What method needs to be implemented in thread class to execute a new thread?
- What method need to be called in Thread class to initiate a new thread?
- Which method can be used to create a delay during the execution of a program?
- What is thread pooling? How is it beneficial?

Persist citizen data in a file using Serialization

Problem

Store the list of citizen data into a file.

Explanation

- Create a new package com.company.serialize
- Create a new class named "Serializer".
- Copy and paste the code below:

```
package com.company.serialization;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

import com.company.collection.CitizenDataManager;

public class Serializer {

    public static void main(String args[]) {
        try {
            FileOutputStream fileOutStream = new FileOutputStream("D:\\citizens.ser");
            ObjectOutputStream objectStream = new ObjectOutputStream(fileOutStream);

            objectStream.writeObject(CitizenDataManager.getCitizenArrayList());

            objectStream.close();
            fileOutStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

- Running the program will result in java.io.NotSerializableException.
- Open. com.company.collection.Citizen
- Import java.io.Serializable
- Modify the class definition as specified below:

```
public class Citizen implements Comparable<Citizen>, Serializable {
```

- Now run the Serializer class.
- Go to D: in windows explorer and check if the file named citizens.ser is created.
- Open the file in Notepad++, which will contain lots of junk characters.
-

Explanation

- Using new keyword, objects were created and stored in the Heap Memory.
- Heap Memory is nothing but the RAM of a desktop or server.
- When JVM exists the java program execution all the Heap Memory is gone and we can't retrieve it any more.
- Serialization helps in storing the state of a java object instance.
- To enable serialization on an object, it has to implement java.io.Serializable interface.
- This interface does not have any methods, hence there is no need to implement any method due to Serializable implementation.

- This kind of an interface is called marker interface, which is just used for marking a capability of a particular class.
- In the example, we are creating a file output stream and object output stream to write serialized content into a file.
- The writeObject() method helps to write a java object into a file.
- Class variables (static variables) are not serialized.
- Instance variables defined as transient are also not serialized.
- Actually, the class details and data of the Citizen ArrayList is stored in this file.
- In this process the Citizen ArrayList and it's data is converted into a series of bytes, so these bytes can also be sent across the network, which is one of the main reason why Serialization was implemented as a functionality in java.

Persist citizen data in a file using Serialization

Problem

Read from a serialized file and convert it into objects.

Solution

- Create a new class Deserializer
- Copy and paste the coding specified below:

```
package com.company.serialization;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;

import com.company.collection.Citizen;

public class Deserializer {

    public static void main(String args[]) {
        try {
            ObjectInputStream objectStream = new ObjectInputStream(
                new FileInputStream("D:\\citizens.ser"));
            ArrayList<Citizen> citizens = (ArrayList<Citizen>) objectStream.readObject();
            System.out.println(citizens);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

- Compile and run the program, check if data displayed is correct or not.

Explanation

- Deserialization process converts bytes to java object instance.
- This can be implemented using FileInputStream and ObjectInputStream.

Understanding Garbage Collection

Activity

Write a program to demonstrate garbage collection.

Solution

- Create a new class Person with attributes name and age with type String and int respectively. Also include a static variable counter to define an unique identifier for each Person object.

```
private static long counter = 0;

private long id;
private String name;
private int age;
```

- Generate constructor.

```
public Person(String name, int age) {
    super();
    this.name = name;
    this.age = age;
}
```

- Source > Override / Implement Methods… > Select "finalize" > Click OK
- Remove the comments and parent method call.
- Include a System.out.println() inside the finalize method.

```
@Override
protected void finalize() throws Throwable {
    System.out.print(this.id + " ");
}
```

- Create a new class GarbageCollector in com.company.garbage package
- Implement the following coding in the main method:

```
Person person = new Person("John", 25);
System.gc();
```

- Execution of GarbageCollector should not result in any output.
- Assign person to null after creation.

```
Person person = new Person("John", 25);
person = null;
System.gc();
```

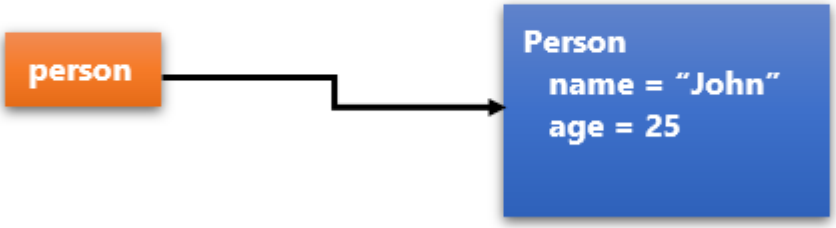
- Execution of GarbageCollector should display 1 as output.

Explanation

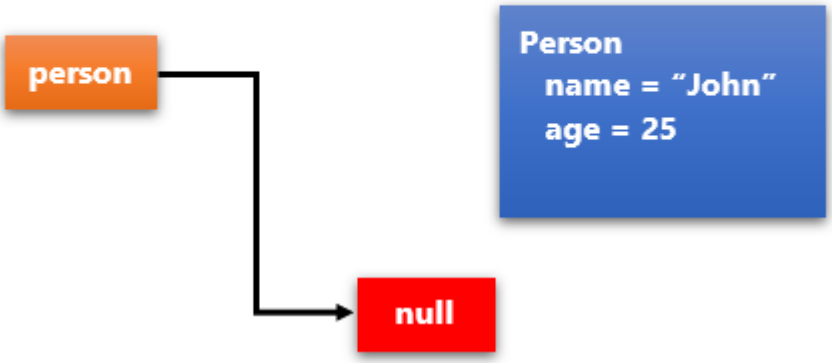
- In java, the developer need not worry about releasing memory that was used.
- For example, we created an array list of Citizen, but we did not worry about clearing this object after usage.
- In C++, programming language, the developer has to write code using delete keyword to remove unwanted objects from memory.
- In java, Garbage Collector is a thread that runs in the background.
- This thread gets executed when the available heap memory is lesser.

- Garbage Collector may get executed in other instances, but we will not know when it will get executed.
- When Garbage Collector executes, it will find all the objects in heap memory which does not have any reference.
- In the above example, when we assigned person to null then the Person object created in the previous line becomes an orphan and does not have any references, which makes it eligible for Garbage Collection. Refer diagram below.

```
Person person = new Person("John", 25);
```




```
person = null;
```



- The gc() method is provided in the System method to initiate Garbage Collections, but we cannot guarantee whether Garbage Collector will execute or not.
- When Garbage Collector removes an object from heap method, Garbage Collector calls the finalize method of the object. If the finalize method is overridden, then the implementation in our class is called.
- When we executed Garbage Collector without assigning person to null, the finalize method was not called.
- After assigning person to null, we can see that the finalize method is called. As per the diagram above the Person object instance does not have any reference, hence it becomes eligible for Garbage Collection.

Solution (making object eligible for Garbage Collection using assignment)

- Include new lines in the main() method as specified below and execute.

```
Person person = new Person("John", 25);
person = null;

Person person1 = new Person("John", 25);
Person person2 = new Person("John", 25);
person1 = person2;

System.gc();
```

- Now the output should display values 1 and 2.
- Modify the code within main() method with the following lines of code:

```
public static void main(String args[]) {

    /*Person person = new Person("John", 25);
    person = null;

    Person person1 = new Person("John", 25);
    Person person2 = new Person("John", 25);
    person1 = person2;*/

    for (int i = 0; i < 10; i++) {
        Person newPerson = new Person("John", 25);
    }

    System.gc();

}
```

- The execution of the above program will not result in any output.
- Modify the 10 in the loop with 1000000 and execute the program. This represents 1 million.
- Modify the Eclipse settings to accommodate the display of the output generated.
- Eclipse > Window > Preferences > Type “console” and make the following changes:
    - Check “Fixed width console”
    - Uncheck “Limit console output”
- Click “Apply and Close”
- Now run GarbageCollector class. This should display numerous numbers.

Explanation

- When person1 is assigned with person2 reference then the object created for person1 will become an object without any reference. Hence it becomes eligible for Garbage Collection.

Solution (making object eligible for Garbage Collection using scope)

- Modify the code within main() method with the following lines of code:

```
public static void main(String args[]) {

    /*Person person = new Person("John", 25);
    person = null;

    Person person1 = new Person("John", 25);
    Person person2 = new Person("John", 25);
    person1 = person2;*/

    for (int i = 0; i < 10; i++) {
        Person newPerson = new Person("John", 25);
    }

    System.gc();

}
```

- The execution of the above program will not result in any output.
- Modify the 10 in the loop with 1000000 and execute the program. This represents 1 million.
- Modify the Eclipse settings to accommodate the display of the output generated.
- Eclipse > Window > Preferences > Type "console" and make the following changes:
  - Check "Fixed width console"
  - Uncheck "Limit console output"
- Click "Apply and Close"
- Now run GarbageCollector class. This should display numerous numbers.

Explanation

- In this example, each time the execution of for loop is completed, the Person object created becomes eligible for Garbage Collection, since the newPerson variable cannot be used outside the for loop.
- If the for loop executes for 50 times, then 50 person objects are created.
- When the for loop execution completes, all the 50 object created are eligible for Garbage Collection.
- The scope of a variable is valid only within a block.
- A block is represented within open and close curly braces.
- Typically this is applicable to the open and close curly braces of if, for, while, do..while and method.
- During execution after completion of close curly braces, whatever variables created during the execution of the block will go out of scope, unless and until the object created is provided as reference to another object through a method call.
- What exactly happened in this example:
  - We are creating Person object 1 million times
  - So, at some point of time it exceeds the heap memory, which triggers the execution of Garbage Collector.
  - The Garbage Collector will be randomly picking objects which does not have any references.
  - There will be lots of Person object created earlier which would have gone out of scope.

- Garbage Collector picks up these items and removes them from memory.
- Before removing from memory, it calls the finalize method which is what we see in the screen as output.

Summary

- In Java, the developer need not take care of clearing objects created in memory.
- The Garbage Collector is a daemon thread that gets started when it senses that memory cleansing is required.
- The Garbage Collector identifies objects that does not have any references and removes them from memory.
- Before removing from memory the finalize() method of the object is called.
- The following are the scenarios from which an object can attain non referenced state.
  - When defining a reference to null
  - When defining a reference variable to another reference variable
  - When a variable goes out of scope
  - Island of Isolation
    - Object 1 refers Object 2
    - Object 2 refers Object 1
    - No other active object refers these two objects.
- The garbage collection can be initiated using System.gc() method invocation, but we cannot guarantee whether Garbage Collection will be initialized or not.

Check your understanding

- What is Garbage Collection?
- Java developers have to take care of clearing the memory. (true / false)
- In Java, what takes care of managing the memory?
- How does Garbage Collector know which object to remove from memory?
- List the scenarios when an object comes to dereferenced state.
- What is the purpose of finalize() method?

Packaging your Java application

Problem

- Your friend had requested you to write a Java program that does a scientific calculation and provides and output.
- You have developed an application to this scientific calculation.
- This application has around 15 java source files in different packages.
- Now you have to send this application to your friend so that he can start using the application.
- How do you share this application to your friend? What options comes to your mind?
- Kindly keep in mind that you friend is not familiar with programming languages.

Solution #1 (send the source files)

- Send the java source files to your friend by mail.
- How will your friend run these programs?
- What software is required to run these programs?
- Do they have to download JDK?
- Do they have to download Eclipse?
- After installing Eclipse what steps have to be done to run the program?
- Is sending source files the right way?

Solution #2 (send the compiled class files)

- From solution #1 we had identified that the process is complicated when sending the source files. We decide to send the class files.
- How do we get the class files?
- For this exercise, we will consider compiling all class files in java-learn.
- Compiling in command prompt.
    - Open command prompt in eclipse-workspace\java-learn\src folder
    - Execute the below command to generate the list of all source files.

```
ace\java-learn>dir /s /B *.java > sources.txt
```

    - Execute the command below to compile.

```
e\java-learn>javac -d d:\bin @sources.txt
```

    - Check the folder d:\bin, which will contain the compiled classes.
- We can also get the compiled classes from the bin folder of the Eclipse "java-learn" project.
- Now we have to send all the compiled classes to your friend, but we have a challenge.
- Each class file is in a separate package folder, how do we send it in such a way the folder structure is retained.
- We can zip the com folder in bin and send it by mail.
- Your friend has received the mail and had extracted the zip content into a folder.
- Now how will your friend run your program.
- Is there any software needs to be installed?
- Is JDK needs to be installed?

- Or is JRE needs to be installed?
- After java installation, what needs to be done, how to execute the program?
- Follow steps below to run the program in command prompt:
    - Let us assume that the zip file is extracted in the folder D:\java, such that com folder is available in this folder.
    - Open command prompt.
    - Execute the command below to set the classpath:

```
>set classpath=D:\java
```

    - If we want to run the AverageCalculator in com.company.collection folder, then we have to run the following command:

```
>java com.company.collection.AverageCalculator
```

- We have to send JRE installation instructions and above instructions to run the program.

Solution #3 (send JAR file)

- Earlier solution was surely an improvement from solution #1. But we can still do better.
- The compiled class files can be put together into a format called JAR.
- JAR is a **J**ava **Ar**chive tool that packages a set of files into a single archive file.
- Options available to create a jar file:
    - Use the export feature in Eclipse to convert the class files into a single jar file.

      Eclipse > Right Click on Project > Export…
    - Use jar command in command line to create a jar file.

      https://www.codejava.net/java-core/tools/using-jar-command-examples
    - Using third party build tools. Some examples provided below:
        - Apache Maven
        - Gradle
        - Apache Ant
- The jar file can be sent to your friend instead of the zip file we created earlier.
- After JRE installation, your friend has to run the following commands to run the program:
    - Setting the classpath, assuming that the JAR file is placed in d:

```
>set classpath=d:\project.jar
```

    - To run AverageCalculator, run the command below:

```
>java com.company.collection.AverageCalculator
```

Additional Information about Jars

- There are lots of open source projects in Java, which provides Java based libraries for various scenarios.
- For example, there is open source project Apache POI, with which we can read or write Microsoft Word, Excel and PowerPoint documents.
- All Java libraries are delivered as a Jar file.
- Go to https://mvnrepository.com in browser.

- Search for “apache commons collection”
- Click on any specific version
- Click on the Jar link to download the specific Jar.
- The downloaded jars can be added in an Eclipse Java project. Refer steps below:
    - Right click on the java project
    - Select “Build Path” > “Configure Build Path…”
    - Click “Add Jars” button and add the downloaded Jar files
- When distributing our jar file to others, we need to send the dependent library jars as well to run.

Reading data from command line

Problem

Modify the com.company.date.TrainingDurationCalculator class to accept Course Start Date and Course Duration as inputs from command line instead of Scanner.

Solution

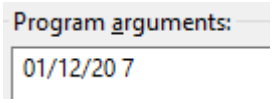
- Open com.company.date.TrainingDurationCalculator in Eclipse.
- Modify the following lines:

```
Scanner scanner = new Scanner(System.in);

System.out.println("Enter course start date in dd/mm/yy format: ");
String input = scanner.nextLine();

System.out.println("Enter course duration in weeks:");
int weeks = scanner.nextInt();
```

- as specified below:

```
String input = args[0];
int weeks = Integer.valueOf(args[1]);
```

- In Eclipse Menu > Run > Run Configuration, which open a new dialog box.
- In this window select TrainingDurationCalculator in the left-hand side, then click the "Arguments" tab in the right-hand side.
- Enter the following value in Program Arguments section.



- Execute this program in command line by running the command below after setting the classpath.

```
>java com.company.date.TrainingDateCalculator 01/12/20 7
```

Solution

- The args[] parameter of main method contains the command line arguments.
- Each command line parameter ris separated by a space.
- The command line parameters are converted into an array and passed as args[].
- Using the array index number, we can directly access those values and use it in our program.

Enum

Explanation

- Use Enum type to represent a fixed set of constants.
- For example, months can be represented as Enum. Refer sample below.

```
public enum Month {
    JAN, FEB, MAR, APR, MAY, JUN,
    JUL, AUG, SEP, OCT, NOV, DEC
}
```

- The enum can be used as specified below:

```
public class EnumTest {
    public static void main(String args[]) {
        Month month = Month.JAN;
        System.out.println(month);

        if (month == Month.JAN) {
            System.out.println("January");
        }
    }
}
```

- Defining a list of constants in this way makes the code more readable.
- Further Reading:

  https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

Nested Classes

Explanation

- Defining a class within a class is called Nested Classes.

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

- Nested classes are divided into two categories:
  - Static Nested Class
  - Inner Class (non-static)

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

- Reasons to use Nested Class:
  - To logically group classes that are used only once.
  - Increases Encapsulation
  - Maintainable Code
- Inner classes can be classified as specified below:
  - Inner Class – Same as the inner class defined above.
  - Local Inner Class – Class defined within a method
  - Anonymous Inner Class
    - Anonymous classes make your code concise.
    - Declare and instantiate a class at the same time.
    - They are Local Classes without a name

```
HelloWorld frenchGreeting = new HelloWorld() {
    String name = "tout le monde";
    public void greet() {
        greetSomeone("tout le monde");
    }
    public void greetSomeone(String someone) {
        name = someone;
        System.out.println("Salut " + name);
    }
};
```

Understanding final keyword

Activity (making a class final)

- Create a new package com.company.finalexample
- Create a new class named Parent and modify the class definition with final.

```
public final class Parent {

}
```

- Create a new class Child which extends from Parent. This fails compilation.

```
public class Child extends Parent {

}
```

The type Child cannot subclass the final class Parent

1 quick fix available:

Remove 'final' modifier of 'Parent'

Press 'F2' for focus

Activity (making a method final)

- Create class com.company.finalexample.FinalMethodParent
- Create a new method with final definition.

```
public final void process() {

}
```

- Create a new class FinalMethodChild in the same package, which extends from FinalMethodParent, then override the process().

```
public void process() {

}
```

Cannot override the final method from FinalMethodDemonstratorParent

1 quick fix available:

Remove 'final' modifier of 'FinalMethodDemonstratorParent.process'(..)

Press 'F2' for focus

Activity (making an instance variable final)

- In class com.company.finalexample.Parent, modify the code as specified below:

```
public final class Parent {
    private static final String CONSTANT = "value";

    public Parent() {
        CONSTANT = "another value";
    }
}
```

The final field Parent.CONSTANT cannot be assigned

1 quick fix available:

Remove 'final' modifier of 'CONSTANT'

Press 'F2' for focus

- It is possible to define the method parameters as final and local variables as well as final.

Reflection API

Explanation

- Reflection is an API in Java
- This is a relatively advanced feature.
- It is used to examine classes, methods and instance variables.
- Scenarios where Reflection API is useful:
  - Develop a framework that will have to instantiate and use classes defined by user. (Example: Spring Framework)
  - There is a need to examine a Java class and display the instance variables and its methods. (Example: Eclipse displays the Java class with member variables and methods ini it in Project Explorer. This is done using reflection)
  - Debugging tools, something like the one in Eclipse needs to examine the data in the class and display the details, which is achieved through reflection.
  - Testing tools can scan through the classes and identify potential methods that can be tested.
- Refer code snippet below:

```
@Test
public void givenObject_whenInvokePublicMethod_thenCorrect() {
    Method sumInstanceMethod
      = Operations.class.getMethod("publicSum", int.class, double.class

    Operations operationsInstance = new Operations();
    Double result
      = (Double) sumInstanceMethod.invoke(operationsInstance, 1, 3);

    assertThat(result, equalTo(4.0));
}
```

Courtesy: https://www.baeldung.com/java-method-reflection

- This code snippet calls publicSum() method in Operations class and gets the result:
  - The method getMethod() gets the reference of a method.
  - The method invoke() calls the respective method and returns the result.
- Further reading:
  - https://docs.oracle.com/javase/tutorial/reflect/index.html
  - https://www.baeldung.com/java-method-reflection

BigDecimal and BigInteger

- **BigDecimal** helps in precision and accuracy of floating-point calculation.

  https://www.geeksforgeeks.org/bigdecimal-class-java/
- **BigInteger** helps in mathematical operations that involve very big calculations that goes beyond the limits of primitive data types.

  https://www.geeksforgeeks.org/biginteger-class-in-java/
- These classes are part of java.math package

Mathematical operations using java.lang.Math

- The java.lang.Math class contains methods for performing basic numeric operations such as:
  - Exponential
  - Logarithm
  - Square Root
  - Trigonometric function
- Reference - https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

StringBuffer and StringBuilder

- String is immutable in Java, which means that once a String object is created the string cannot be modified.
- Hence String manipulations results in creation of numerous new String objects.
- New String objects are created disregarding the older objects created.
- This results in increased heap memory size.
- The StringBuffer and StringBuilder classes help address this issue.
- StringBuffer is thread-safe
- StringBuilder is not thread-safe
- The following methods helps in manipulating string:
  - substring()
  - insert()
  - append()
  - delete()

Lambda Expressions

- New feature introduced in Java 8
- A Lambda Expression is an anonymous function, meaning a function without a name.
- Syntax for defining a Lambda Expression.

```
//Syntax of lambda expression
(parameter_list) -> {function_body}
```

- The following code snippet:

```
b.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e){
      System.out.println("Hello World!");
   }
});
```

- can be written as:

```
b.addActionListener(e -> System.out.println("Hello World!"));
```

- In the first code snippet a new class ActionListener is defined and passed as parameter.
- When using Lambda Expression we are passing a method definition itself as a parameter.
- Iterating an ArrayList and displaying each item can be now implemented in a single line.

```
import java.util.*;
public class Example{
    public static void main(String[] args) {
       List<String> list=new ArrayList<String>();
       list.add("Rick");
       list.add("Negan");
       list.add("Daryl");
       list.add("Glenn");
       list.add("Carl");
       list.forEach(
           // lambda expression
           (names)->System.out.println(names)
       );
    }
}
```

Courtesy: https://beginnersbook.com/2017/10/java-lambda-expressions-tutorial-with-examples/

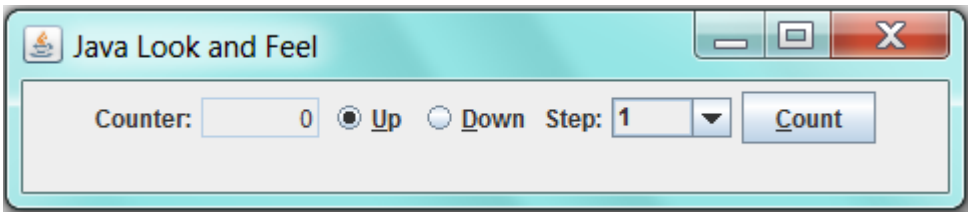
- Further Reading
  - https://beginnersbook.com/2017/10/java-lambda-expressions-tutorial-with-examples/
  - https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Graphical User Interfaces

- **Applet** technology allows to run Java programs within browser. This technology was very popular during the advent of internet. This technology is no more used now.

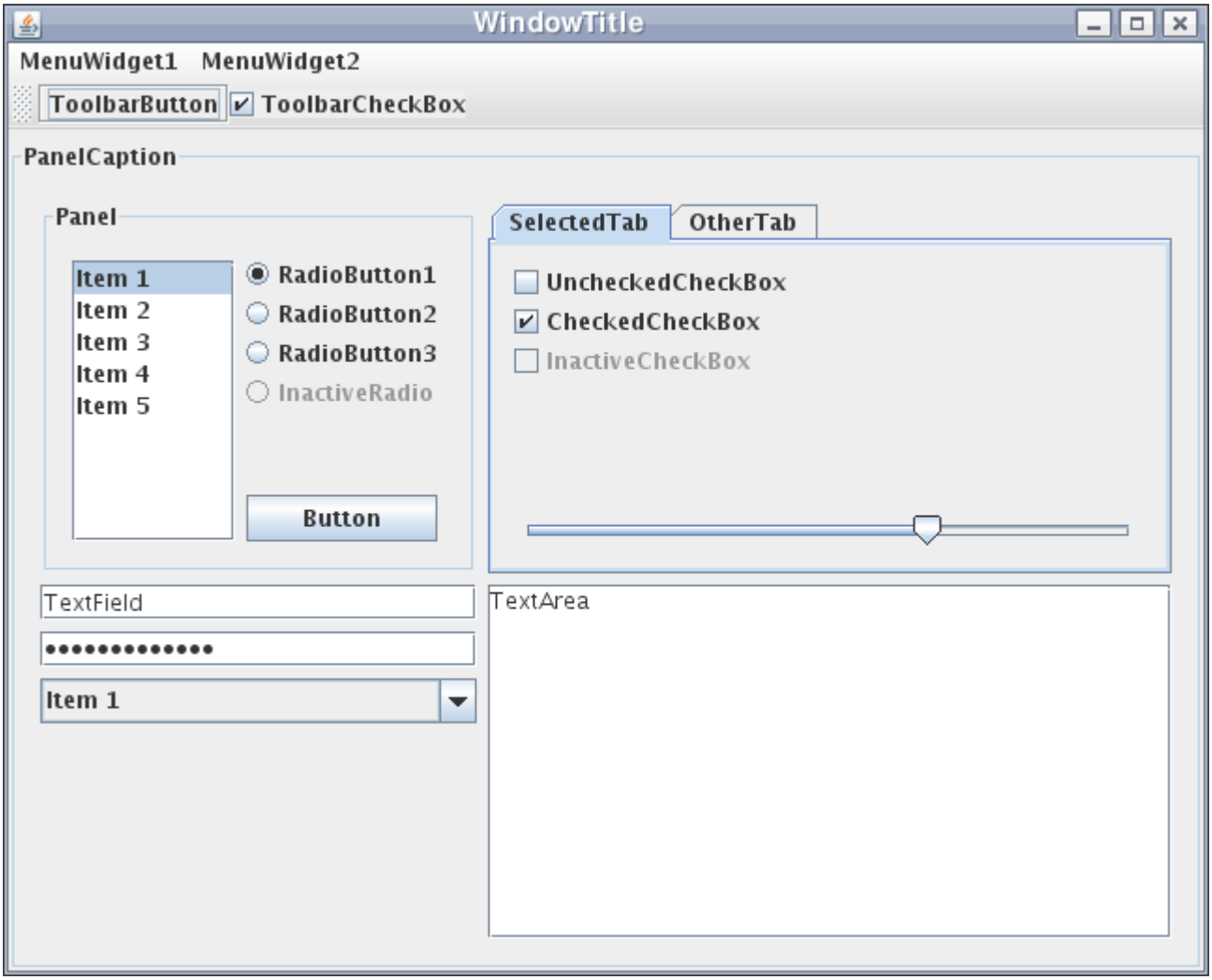
  https://en.wikipedia.org/wiki/Java_applet

- **Java Advanced Window Toolkit (AWT)**
  - Platform dependent API for creating Graphical User Interface
  - Heavy Weight
  - Sample screen developed using AWT.



Courtesy: https://www.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

- **Swing**
  - Swing is a GUI Widget Toolkit for Java
  - Provides more sophisticated set of GUI components than AWT
  - It has built in components like tabbed panel, scroll panes, trees, table, lists, which is not available in AWT.
  - Swing is platform independent.



Key Features of Java

Java is Simple

Java programming language is easy to learn. Java code is easy to read and write.

Java is Familiar

Java reuses lots of syntax from C/C++, but it removes some complexities of C/C++, for example pointers and multiple inheritances are removed. If you are familiar with C++, then it makes it familiar to learn.

Java is Object Oriented

Java supports encapsulation, abstraction, inheritance and polymorphism.

Java is Robust

- Robustness represents the ability to handle errors during execution
- The following aspects make Java Robust:
  - Strong memory management
  - Absence of pointers (No direct access to system memory)
  - Type-checking
  - Exception Handling

Java is Secure

- Static type-checking during compilation
- Runtime checking using security manager. It runs in a sandbox that prevents doing any harm to the system in which Java is running.

Java is High Performance

- The compiled byte code is highly optimized
- This makes Java run in full speed
- In addition, computation intensive code can be re-written into native code using Java Native Interface (JNI)

Java is Multithreaded

- Java has built-in multithreading capability
- It is possible to build applications with many concurrent threads

Java is Platform Independent

The compilation to byte-code enables a Java program to be compiled once and can run in any operating system.

Interview Questions

https://beginnersbook.com/2013/05/java-interview-questions/

Java Cheat Sheet

https://introcs.cs.princeton.edu/java/11cheatsheet/