Introduction

- Objectives:
  - Organizing the data for processing is a computer program is a critical step in the solution of a problem.
  - Selecting the best data organization strongly affect the performance of the computer program solution
  - Most programming languages offer a set of primitive (basic) data types and ways to build complex data types.

- **Basic definitions.**
  - A data type is a set of values and a set of operations defined on those values:
    - Values
    - Operations

  - For example, the values of the primitive data type int are integers between -2^{31} and $2^{31} - 1$;

  - The operations of int are the basic arithmetic and logical operations.

  - **Java**:
    - Programming in Java is largely based on doing so with data types known as reference types.
    - Java has eight primitive data types: boolean, byte, char, double, float, int, long, and short.

- **Java Basic Data Types:**
  - **byte:**
    - The byte data type is an 8-bit signed two's complement integer.
    - It has a minimum value of -128 and a maximum value of 127 (inclusive).
    - The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters.
  - **short:**
    - The short data type is a 16-bit signed two's complement integer.
    - It has a minimum value of -2^{15}=-32,768 and a maximum value of 2^{15}= 32,767 (inclusive).
    - Use short to save memory in large arrays, in situations where the memory savings actually matters.
  - **int:**
    - The int data type is a 32-bit signed two's complement integer.
    - It has a minimum value of
      -2^{15} = -2,147,483,648 and a maximum value of
      -2^{15} = 2,147,483,647 (inclusive).
    - It is the default for integral values,
  - **long:**
    - The long data type is a 64-bit signed two's complement integer.
    - It has a minimum value of
      -2^{63} = -9,223,372,036,854,775,808
      and a maximum value of

2^{63} = 9,223,372,036,854,775,807 (inclusive).

- Use this data type when you need a range of values wider than those provided by int.

- **float:**
  - The float data type is a single-precision 32-bit IEEE 754 floating point.
  - Use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- **double:**
  - The double data type is a double-precision 64-bit IEEE 754 floating point.
  - It is the default data type for decimal values

- **boolean:**
  - The boolean data type has only two possible values: true and false.
  - Use this data type for simple flags that track true/false conditions.
  - This data type represents one bit of information
- **char:**
  - The char data type is a single 16-bit Unicode character.
  - It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

- Java Data Type **Default Values**:

Data Type	**Default Value**
Byte	0
Short	0
Int	0
Long	0L
Float	0.0f
Double	0.0dchar'\u0000'
String (or any object)	null
Boolean	false

  - Bad programming style: Do not depend on default values.

- Data Structure:
  - A means of storing a collection of data
  - It is the requirement of the elements of the structure, the relationships between them, and the operations that may be performed on them
  - The choice of a data structure can affect the performance of a solution: slow vs. fast
  - **Data Structure Taxonomy:**
    - Linear and non-linear data structures
    - Homogenous and non-homogenous data structures
    - Primitive and non-primitive data structures

  - **Primitive and Non-primitive Data Structures**
    - Primitive Data Structures
      - Are primitive types
      - Integer, float
    - Non-primitive Data Structures
      - Composite data structures
      - Array, records

  - **Linear and Non-linear Data Structures**
    - Linear Data Structures
      - Data elements are arranged in a linear sequence
      - Examples:
        - Array
        - Linked List
        - Queue
        - Stack
    - Non-linear Data Structures:
      - Data elements are not stored in a sequence

        - Examples:
            - Tree
            - Graph
            - Forest

    - **Homogeneous and Non-homogeneous Data Structures**
        - Homogenous Data Structures:
            - Data elements are of the same type:
            - Example:
                - Array
        - Non-homogenous Data Structures:
            - Data elements are of different types
            - Examples:
                - Records or Structure

    - **Data structures in Computer Science:**
        - Use a computer to efficiently solve a problem:
            - Understand the requirements of the problem
            - Implementing these requirements with a computer program
            - Computer programs consist of the following:
            - Computer Program = Algorithm + Data Structure

- Algorithm:
    - To implement the business logic of the problem
    - It is a concise list of steps to solve a problem
    - There are more than one algorithm for a problem:
        - Choose an efficient algorithm
    - How to choose the best algorithm?
        - Performance Analysis.

- **Developing Your Application:**
  - **Preconditions and Postcondions**
    - Increase the reliability of your application.
    - They are a method of specifying what a function accomplishes.
    - Frequently a programmer must communicate precisely what a function accomplishes, without any indication of how the function does its work.
    - What Preconditions and Postconditions?
    - The **precondition** statement:
      - Indicates what must be true before the function is called.
      - The programmer who calls the function is responsible for ensuring that the precondition is valid when the function is called.
    - The **postcondition** statement
      - Indicates what will be true when the function finishes its work.
      - The programmer who writes the function counts on the precondition being valid, and ensures that the postcondition becomes true at the function's end.
    - **Example:**
      void write_sqrt(double x)
      // Precondition: x >= 0.
      // Postcondition: The square root of x has

// **been written to the standard output.**

- In this example, the precondition requires that: **x >= 0** be true whenever the function is called.

- The postcondition always indicates what work the function has accomplished. In this case, when the function returns the square root of **x** has been written.

- Another Example:

```
bool is_vowel( char letter )
// Precondition:  letter is an uppercase or
// lowercase letter (in the range 'A' ... 'Z' or 'a' ... 'z') .
// Postcondition: The value returned by the
// function is true if Letter is a vowel;
// otherwise the value returned by the function is
// false.
```

- **What values will be returned by these function calls?**

```
is_vowel( 'A' );
is_vowel(' Z' );
is_vowel( '?' );
```

- In Java:
  - Uses **assert** statement to implement both preconditions and postconditions.
  - Enable Assertions:
    - Command Line::
      - java -enableassertions ClassName
    - IDE:
      - You need to enable assertion in your IDE(e.g., Eclipse)
      - In Eclipse:
        - Windows -> Preferences -> Java -> Installed JREs.
        - Select your JDK and
        - Click the Edit… button.
        - In the "Default VM Arguments" box, add **-ea.**
  - **Example:**

```
import java.util.*;
public class Precondition {
        public static void main( String args[] ) {
                int value;
          Scanner scanner = new Scanner( System.in );

          System.out.print( "Enter a non-negative value: " );
          value = scanner.nextInt();
          assert( value >= 0 ) :  "Your input should be positive: " +
        value;
          System.out.println( "You have entered: " + value );
          System.out.println( "\nThe square root of your input is: " +
        Math.sqrt(value) );
        }
 }
```

- **Testing and Debugging**
  - Is Compile-Run-Generate-Outputs enough?
  - Required Testing!!!
  - Which test data?
    - You must know which output a correct program should produce for each test data.
    - The test inputs should include those inputs that are most likely to cause errors.
  - Tips for Test data:
    - Boundary values
    - Fully Exercising Code: make sure each line of your code is executed.
    - Values of variables: negative, zero, positive, Expecting a numbers or strings, etc.