Lecture 1

Features of Java

- Object Oriented (Everything in Java is an object)
- Simple (Syntax is similar to C++)
- Secure
  - Languages like C++ uses Runtime environment of operating systems
  - Java uses its own Runtime environment
- Dynamic
  - Java supports the dynamic loading of classes.
  - It also supports functions from its native languages, i.e., C and C++.

- Multi-threaded
  - A thread is like a separate program, executing concurrently.
  - Threads share a common memory area.
- Portable
  - Java bytecode can be run on any platform
- Architecture-neutral
  - No implementation dependent features, for example, the size of primitive types is fixed.
  - In C, for 32 bit architecture, **int** data type takes memory of 2 bytes whereas 64 bit machine takes memory space of 4 bytes.
  - In Java, **int** takes memory of size 4 bytes for both types of machines.

- Platform Independent
  - Java Runtime Environment (JRE) and Java Virtual Machine (JVM)
  - JRE is one of three Java platform components that are required for any Java program to run successfully.
    - Other two are: Java Development Kit (JDK) and Java Virtual Machine (JVM)
  - JRE combines the Java code created using the JDK with additional built-in libraries.
    - It creates a JVM instance.
  - JVM is an interpreter that runs the Java program line by line
  - JRE facilitates platform independence for Java applications.
    - You can write them once and run them anywhere.

- JDK is a software layer above the JRE which contains the compiler, debugger and other tools for software development.
- JDK compiles the code and generates a Bytecode which is passed to the JRE.
- JRE contains class libraries, supporting files, and the JVM which is used to run the byte code on any device.

- JRE uses three core components to work:
  - **ClassLoader**
  - Java class libraries contain collections of pre-written code that can be used as needed.
  - The Java ClassLoader dynamically loads all class files necessary into the Java Virtual Machine (JVM) on demand.
  - **Bytecode verifier**
  - They bytecode verifier in the JRE checks the format and accuracy of the Java code before loading it into the JVM.
    - For example, if the code violates system integrity or access rights, the JRE will not load the class file.
  - **Interpreter**
  - After the bytecode successfully loads, the Java interpreter creates the JVM instance that runs the Java program on the underlying machine.

First Program

```
class Person{

//statements

//creating main method→ entry and exit point of any
Java program

 public static void main(String args[]){
  System.out.println(“Hello World”);
 }
}
```

Basic concepts of Object Orientated Programming (OOP) :

- **Object/Class**
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

- Class: a group of entities which have common properties.
  - It is a template or blueprint from which objects are created.
- Object: An object is an instance of a class.
- Syntax for class:

```
class <class_name>{
    Data member;
    method;
}
```

```
class Person{
int id; //data member
String name;
 //creating main method→ entry and exit point of any Java program
 public static void main(String args[]){
  //Creating an object
  Person p=new Person();
  //Printing values of the object
  System.out.println(p.id);//accessing member with reference variable
  System.out.println(p.name);
 }
}
```

Lecture 2

Previous Lecture Summary

- Java Features:
  - A high level object oriented programming level
  - Portable
  - Secure
  - Robust memory management
  - Multi-threaded
  - Dynamic (classes loaded on demand)
  - Platform independent
- Java was created by James Gosling at Sun Microsystems (now acquired by Oracle Corporation) in 1995

- Java Supports Object oriented design paradigm
  - Abstraction
  - Encapsulation
  - Inheritence
  - Polymorphism

- Java Program consists of creation of classes and their instantiations (Object)

Abstraction

- A Design which hides the details of how something works while still allowing the user to access complex functionality
- How do we accomplish abstraction in Java?
  - Using Classes
- Class: Informally, a collection of similar entities grouped together with associated functions.
  - Formally, It defines a new data type for our program.
  - Similar to Stuct in C/C++ (group together different types of information)
  - Difference between two?

- In a **struct** members are can be accessed from outside a class
- In a **class** members are made accessible only from inside the class implementation
- Class has two parts:
  - **Interface:** specifies what operations can be performed on instances of the class (the abstraction boundary)
  - an **implementation:** specifies how those operations are to be performed
- Encapsulation: The process of grouping/binding related information (Member variables) and relevant functions (Member functions or methods) into one unit and defining where that information is accessible.
  - A java class is an example

- Classes as a blueprint for objects
  - A blueprint describes a general structure, and we can create specific **instances** of our class using this structure.
  - Instance: When we create an object that is our new type, we call this creating an instance of our class
  - E.g.: Stack<Integer> new_stack = new Stack<Integer>();
    - An Object (instance of class Stack) of type Stack.
  - E.g. Vector<Integer> v = new Vector<Integer>(n);

```
class Person{
    private String name;
    private int id;
    public void setName(String n){          name=n;        }
    public String getName(){          return name;    }
    public void setID(int ID){          id=ID;   }
    public int getID(){          return id;          }
    }
class Person_Class{
    public static void main(String [] args){
        Person first=new Person();
        first.setID(1);
        first.setName("first name");
        System.out.println(first.getName());       }
}
```

How to design Classes?

- There are three main parts:
  - Member variables:
    - Variables stored within the class
    - Usually not accessible outside the class implementation
    - What other variables make up this new variable type (Class)?
  - Member Functions:
    - Functions that can be called by an object of the class
    - e.g. new_stack.push(5), new_stack.pop() etc.
    - What functions can you call on a variable of this type?
  - Constructor
    - Special Function which get called when an object is created
    - e.g. Stack <Integer> new_stack;
    - What happens when you make a new instance of this type?

Basics of Java

Basic Java Syntax

- Primitive Types and Variables
  - Example: boolean (1 bit), char (2 bytes), byte (1 byte), short (2 bytes), int (4 bytes), float (4 bytes), double (8 bytes) etc.
  - These types are not objects
  - It signifies that we do not use **new** operator to create a primitive variable.
  - Declaration:
    - int val1=2;
    - double val2=2.34;
  - Java sets primitive variables to 0 or false in case of Boolean variable.
  - All object references are initially set to **null.**

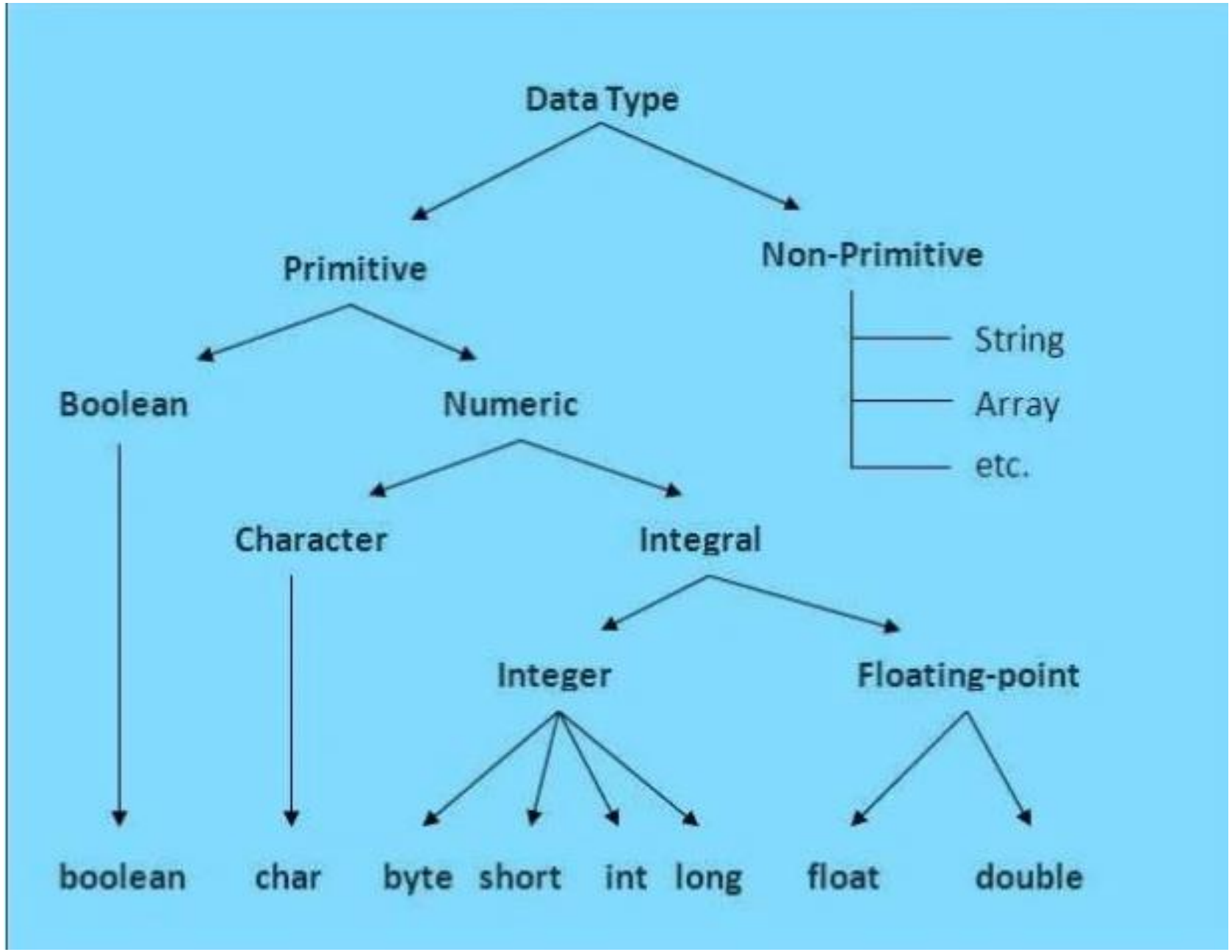

Image Credits: https://www.slideshare.net/MahikaMotwani1/core-java-tutorials-by-mahika-tutorials

Let us check

- int i=1.2
  - Compiler error
- float fVar=1.2;
  - No error
- boolean flag=1;
  - Compiler error
- Double d=8.3
  - Compiler error

- All Java assignments are right associative
  - int a=1, b=2,c=3;

  a=b=c;
- Example:

```
int a=1,b=2,c=3;
a=b=c;
System.out.println("a="+a+"b="+b+"c="+c);
```

Output?

a=3b=3c=3

- Basic Mathematical Operators: *, /, %, + , -
  - *, /, % have higher precedence than + or –
- Statement: int val=2;
  - Block of statement (compound statement enclosed in curly braces) e.g., {int val=2; int ans=3;}
  - Blocks may contain other blocks.
- Flow of control
  - **Alternation**: if, if-else, switch
  - **Loop**: for, while, do-while
  - **Escapes**: break, continue, return

- **if, if else,** nested **if else**
  - One have to be cautious of nested if-else (should use curly braces for each nesting)
- **break** statement causes an exit from the inner **while**, **do**, **for** or **switch** statement.
- **continue** statement can be used only with the **while**, **do** or **for** loops
  - Causes the inner most loop to start the next iteration immediately

Arrays

- List of similar data type
  - Data type can be a custom class: hence an array of objects
- Has fixed name, type and length
  - Must be declared when array is created
  - During the execution of the code these values cannot be changed
- Array are accessed by their indices, starting with **index 0.**
- Declaration: int newArray[]; declares newArray to be an integer array.
  - newArray=**new** int[8]**;**
- Another way to declare: int newArray[]=**new** int[8]**;**

- For loops can be used to loop through the array elements.
- Array of objects:
  - Objects of class Person
  - Person listOfPersons[]=new Person[5]; // sets up 5 memory spaces which can hold references to 5 Person type objects.
    - listOfPersons[0]=new Person(…);