CS205 Object Oriented Programming in Java

Module 2 - **Core Java Fundamentals (Part 1)**

Prepared by Renetha J.B.

Topics

- Core Java Fundamentals:

✓ **Primitive Data types**

- ✓ **Integers**
- ✓ **Floating Point Types**
- ✓ **Characters**
- ✓ **Boolean**

Introduction

- Most fundamental elements of Java:
  - data types
  - variables
  - arrays

Introduction(contd.)

- **Java Is a Strongly Typed Language**
  - First, every **variable** has a **type**, every **expression** has a **type**, and every **type is strictly defined**.
  - Second, **all assignments**, whether explicit or via parameter passing in method calls, are **checked for type compatibility**.
  - **No automatic coercions or conversions** of conflicting types .
  - The Java compiler checks all expressions and parameters to ensure that the types are compatible.
  - Any type mismatches are errors that must be corrected before the compiler will finish compiling the class

The Primitive Types

- The primitive types are also commonly referred to as ***simple types.***

- The primitive types represent **single values**—not complex objects

The Primitive Types(contd.)

Java defines eight *primitive types of data:*

- **byte**
- **short**
- **int**
- **long**

- **float**
- **double**

- **char**

- **boolean**

The Primitive Types(contd.)

Java defines eight *primitive types of data- **FOUR GROUPS**:*

- **byte**
- **short**
- **int**
- **long**

→ **INTEGERS**

- **float**
- **double**

→ **FLOATING-POINT NUMBERS**

- **char** → **CHARACTERS**

- **boolean** → **BOOLEAN**

Primitive Types -four groups

- **Integers** This group includes **byte, short, int, and long,** which are for **whole-valued signed** numbers.
- **Floating-point numbers** This group includes **float and double,** which represent **numbers with fractional precision**.
- **Characters** This group includes **char, which represents symbols in a character set,** like **letters** and **numbers**.
- **Boolean** This group includes **boolean, which is a special type for representing true / false** values.

Integers

- Java defines four integer types:
  - **byte**
  - **short**
  - **int**
  - **long**
- Can be **signed, positive or negative values.**
- Java *does not support unsigned*, positive-only integers.
- The **width** of an integer type is not the amount of storage it consumes, but it is the behavior it defines for variables and expressions of that type

Integers

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

byte

- The smallest integer type is byte.
- This is a signed 8-bit type
- It has a range from –128 to 127.
- Useful when working with a stream of data from a network or file.
- E.g. **declares** two byte variables called b and c:

byte b, c;

short

- short is a signed 16-bit type.
- It has a range from –32,768 to 32,767.
- It is the least-used Java type.
- Examples of **short variable declarations:**

```
short s;
short t;
```

int

- Variables of type **int are** commonly employed
  - to control loops
  - to index arrays.
- When **byte** and **short** values are ***used in an expression*** they are promoted to int when the expression is evaluated.
- int is often **the best choice** when an integer is needed.

long

- long is a signed 64-bit type and is useful for those occasions where an *int type is not large enough* to hold the desired value.
- The range of a long is **quite large**.

Floating-Point Types

- Floating-point numbers, also known as **real numbers.**
- They are used when evaluating expressions that require fractional precision.

Name	**Width in Bits**	**Approximate Range**
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

float

- The type float specifies a **single-precision value** that uses **32 bits** of storage.
- Single precision is faster on some processors and **takes half as much space as double precision,** but will become **imprecise** when the values are either very large or very small.
- Variables of type float are useful when you need a fractional component, but **don't require a large degree of precision**.
- Example **float variable declarations:**

float hightemp, lowtemp;

double

- Double precision, as denoted by the **double** keyword, uses 64 bits to store a value.
- Double precision **is actually faster** than single precision on some modern processors.
- math functions, such as **sin( ), cos( ), and sqrt( ),** return **double** values.

E.g. double

```
// Compute the area of a circle.
class Area {
    public static void main(String args[])
    {
     double pi, r, a;
     r = 10.8;
     pi = 3.1416;
     a = pi * r * r;
     System.out.println("Area of circle is " + a);
    }
}
```

OUTPUT

Area of circle is 366.436224

Characters

- In Java, the data type used to **store characters** is **char.**
- char in Java is **not the same** as char in C or C++.
  - In C/C++, char is 8 bits wide.
- Java uses **Unicode** to represent characters.
- Unicode defines a **fully international character set** that can represent all of the characters found in all human languages.
- So it requires 16 bits.
- The range of a char is **0 to 65,536.**
- There are **no negative chars**

```
char ch1='a';
```

```
// Demonstrate char data type.
class CharDemo
{
    public static void main(String args[])
    {
    char ch1, ch2;
    ch1 = 88;              // code for X
    ch2 = 'Y';
    System.out.print("ch1 and ch2: ");
    System.out.println(ch1 + " " + ch2);
    }
}
```

OUTPUT

```
ch1 and ch2:X Y
```

char act as integer type -arithmetic operations

```
// char variables behave like integers.
class CharDemo2
{
    public static void main(String args[])
    {
    char ch1;
    ch1 = 'X';
    System.out.println("ch1 contains " + ch1);
    ch1++;                    // increment ch1
    System.out.println("ch1 is now " + ch1);
    }
}
```

OUTPUT

```
ch1 contains X
ch1 is now Y
```

Booleans

- Java has a primitive type, called **boolean**, for **logical values.**
- It can have only one of two possible **values**, **true** or **false**.
- This is the **type returned by all relational operators**,
  - boolean is also the type required by the conditional expressions that govern the control statements such as if and for.

```
// Demonstrate boolean values.
class BoolTest
{
    public static void main(String args[]) {
    boolean b;
    b = false;
    System.out.println("b is " + b);
    b = true;
    System.out.println("b is " + b);
    if(b)
        System.out.println("This is executed.");
    b = false;
    if(b)
        System.out.println("This is not executed.");
    System.out.println("10 > 9 is " + (10 > 9));
    } }
```

OUTPUT

```
b is false
b is true
This is executed.
10 > 9 is true
```

Reference

- Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.