Java Syntax

Lecture - 2

Tokey Ahmmed

Lecturer, Dept. of CSE

Varendra University

tokey@vu.edu.bd

Java Program File

- Java program/application **begins with a class name**, and that class must match the filename.
- Java files are saved in `.java` file format.

```
public class FirstProgram {

  public static void main(String[] args) {

     //This will print Hello world:
     System.out.println("Hello World");

  }

}
```

FirstProgram.java

Understanding Code Structure

Java Class

- All the java script is be a java class.
- Every line of code that runs in Java must be inside the class.
- { … } left brace and right brace define the body of the class .

```
public class FirstProgram {

  public static void main(String[] args) {

     //This will print Hello world:
     System.out.println("Hello World");

  }

}
```

Main Method

- In Java programs, the point from where the program starts its execution or simply the entry point of Java programs is the main() method.

Public: It is an *Access modifier*, which makes the *main()* method globally available, so that JVM can invoke it from outside the class.

Static: It is a *keyword makes* The *main()* method static so that JVM can invoke it without instantiating the class.

```
public class FirstProgram {

  public static void main(String[] args) {

     //This will print Hello world:
     System.out.println("Hello World");

  }

}
```

Void: It is a keyword and used to specify that a method doesn't return anything. it doesn't make any sense to return from main() method as JVM can't do anything with the return value of it.

String[] args: It stores Java *command line arguments* in a array of type *java.lang.String* class.

{ ... } left brace and right brace define the body of the function.

Comment

The Java comments are the statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation.

❑2 types of comments in Java:

1. **Single Line Comment:**

   // This is a comment

2. **Multi Line Comment:**

   /* This is a example of

   multiline comment*/

```java
public class FirstProgram {

  public static void main(String[] args) {

     //This will print Hello world:
     System.out.println("Hello World");

  }

}
```

Print Function

- println() is a method that prints the argument passed to it with a newline.

System: It is a final class defined in the java.lang package.

out: This is an instance of PrintStream type, which is a public and static member field of the System class.

```
public class FirstProgram {

  public static void main(String[] args) {

     //This will print Hello world:
     System.out.println("Hello World");

  }

}
```

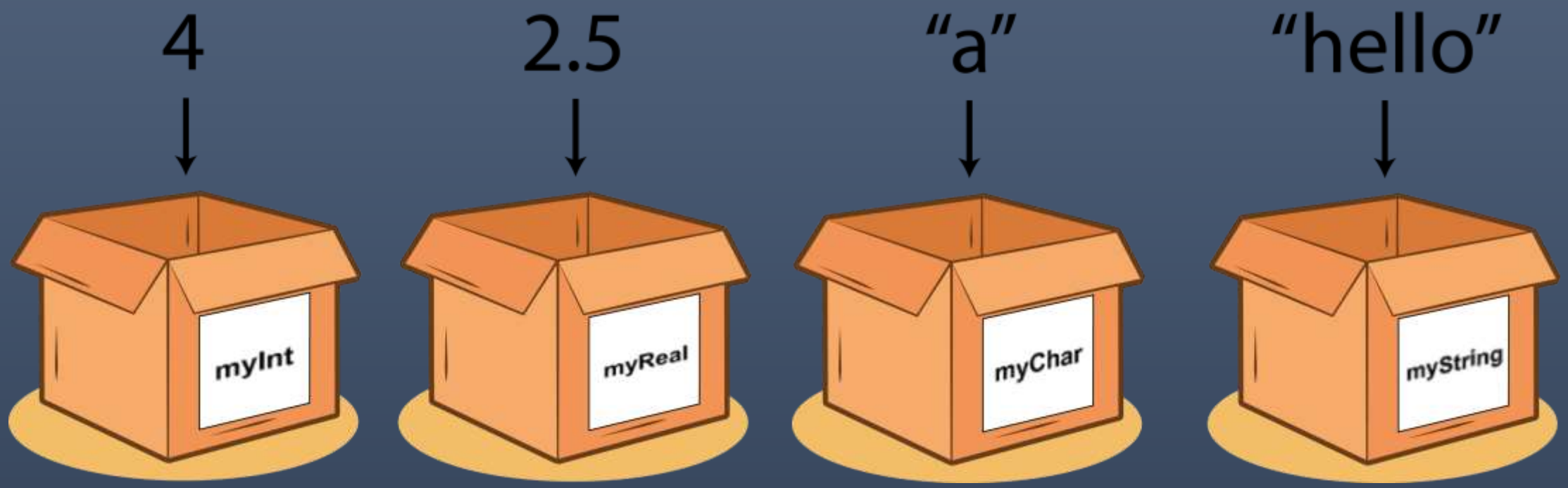
Print Escape Sequence

- A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

Variable

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location.



Variable

- There are three types of variables in java: local, instance and static.

1) Local Variable: A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
A local variable cannot be defined with "static" keyword.

2) Instance Variable: A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable: A variable which is declared as static. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.
No need to create instance for accessing static variable.

Instance variables Example:

```
public class Employee {
     // this instance variable is visible for any child class.
     private String name;

     // salary variable is visible in Employee class only
     private double salary;

     // The name variable is assigned in the constructor
     public Employee(String empName) {
          name = empName;
     }

     // The salary variable is assigned a value.
     public void setSalary(double empSal) {
          salary = empSal;
     }
     // This method prints the employee details.
     public void printEmp() {
          System.out.println("name : " + name);
          System.out.println("salary :" + salary);
     }

     public static void main(String args[]) {
          Employee empOne = new Employee(“Tokey");
          empOne.setSalary(1000);
          empOne.printEmp();
     }
}
```

Output:

name : Tokey

salary :1000.0

Class/static variables Example:

```
public class Employee{

// salary  variable is a private static variable
private static double salary;

// DEPARTMENT is a constant
public static final String DEPARTMENT = "Development ";

public static void main(String args[]){
        salary = 1000;
        System.out.println(DEPARTMENT+"average salary:"+salary);
    }
}
```

Output: Development average salary:1000

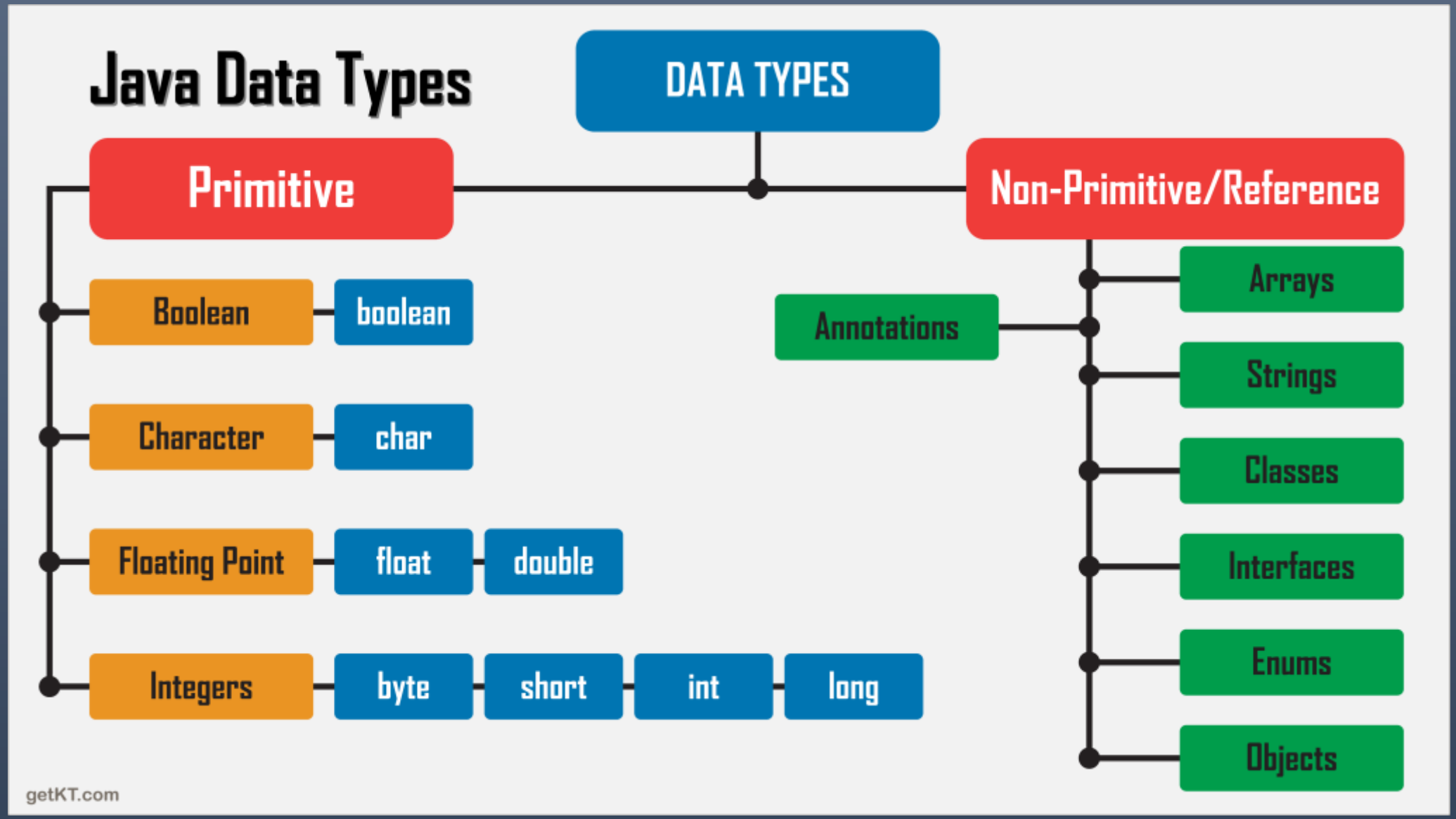
Note: If the variables are access from an outside class the constant should be accessed as Employee.DEPARTMENT

Scope of Variable/Objects

- Java Variable/objects don’t have the same lifetimes as primitives.
- When you create a Java object using **new**, it hangs around past the end of the scope.
- Here, the scope of name scope1 is delimited by the {}scope1 but the String object hangs around until closing brace.

```
{
   String name = new String(“Tokey Ahmmed");
} /* end of scope*/
```

Data Type


Java Data Types
DATA TYPES
Primitive
Non-Primitive/Reference
Boolean
boolean
Character
char
Floating Point
float
double
Integers
byte
short
int
long
Annotations
Arrays
Strings
Classes
Interfaces
Enums
Objects
getKT.com

Primitive Data Type

Name	Default Value	Size	Type
byte	0	1 byte	Integral Value
short	0	2 byte	Integral Value
int	0	4 byte	Integral Value
long	0	8 byte	Integral Value
float	0.0f	4 byte	Floating Point
double	0.0d	8 byte	Floating Point
char	'\u0000' (means 0 in ASCII)	2 byte	Character
boolean	false	1 bit	Boolean

Identifiers

- In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label.

For Example:

- Test : class name.
- main : method name.
- String : predefined class name.
- args : variable name.
- a : variable name.

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 20;
    }
}
```

Identifiers

- Rules for defining Java Identifiers

- There are certain rules for defining a valid java identifier. These rules must be followed, otherwise we get compile-time error. These rules are also valid for other languages like C,C++.

- The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), '$'(dollar sign) and '_' (underscore).For example "geek@" is not a valid java identifier as it contain '@' special character.
- Identifiers should not start with digits([0-9]). For example "123geeks" is a not a valid java identifier.
- Java identifiers are case-sensitive.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- Reserved Words can't be used as an identifier. For example "int while = 20;" is an invalid statement as while is a reserved word. There are 53 reserved words in Java.

Java Modifiers:

- There are two categories of modifiers:
- **Access Modifiers:** default, public, protected, private
- **Non-access Modifiers:** final, abstract

Java Modifiers:

Access Control Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

- The four access levels are:
- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality.

- The static modifier for creating class methods and variables
- The final modifier for finalizing the implementations of classes, methods, and variables.
- The abstract modifier for creating abstract classes and methods.
- The synchronized and volatile modifiers, which are used for threads.

User Input

- Java **Scanner class** allows the user to take **input from the console**.
- It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, and byte.
- To use the Scanner class, **create an object of the class** and use any of the available methods found in the Scanner class.

❑Syntax:

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

User Input

```
public class  MyFirstJavaProgram{
    public static void main (String args[]) {
        int id;
        String name;

        Scanner sc=new Scanner(System.in);

        System.out.print(“Dear user, please ID, Name:”);
        id=sc.nextInt();
        name=sc.nextLine();

     } // end of main
} // end of class
```

User Input

- Java Scanner class provides the following methods to read different primitives types:

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

Java Operators

- Java provides a rich set of operators to manipulate variables.
- We can divide all the Java operators into the following groups –

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Assignment Operators

- These are Abbreviate assignment expressions
- Any statement of form
  - *variable* **=** *variable* <*operator*> *expression* **;**
- Can be written as
  - *variable operator***=** *expression* **;**

❑For example, addition assignment operator **+=**

- **c = c + 3**
- can be written as: **c += 3**

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Increment/Decrement Operators

- Unary increment operator (**++**)
  - Increment variable's value by **1**
- Unary decrement operator (**--**)
  - Decrement variable's value by **1**

- Pre-increment / pre-decrement operator
- Post-increment / post-decrement operator

Increment/Decrement Operators

Operator	Called	Sample expression	Explanation
++	Pre-increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	Post-increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	Pre-decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	Post-decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.