Yasir Ahmad
Visiting Faculty Member
From
ICS & IT Department
The University Of Agriculture Peshawar

Programming Languages II -- Java

An Introduction to
Programming in Java

About the Course

- You should already:
  - be a competent programmer in a programming language like C, C++ etc.
  - have a good understanding of object oriented programming

Course Contents

- Introduction to Programming in Java
- Object Oriented Programming in Java
- Java Threads
- File Handling
- GUI
- Other Topics
  - Servlets
  - Exceptions

Resources

- Online Resources
  - http://docs.oracle.com/javase/tutorial/java/TOC.html
  - The Java Language Specification, Java SE 7 Edition
  - The Java Virtual Machine Specification, Java SE 7 Edition (mostly for further reading)
- Java How to Program: Early Objects Version (8th Edition) by Paul and Harvey Deitel
- Other books – look for Java 1.5 (at least) or later

Java Hello World

```
/* This is a hello world example in Java
   that will simply display Hello World
   on the monitor */

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

An idiom(محاورے) explained

You will see the following line of code often:

public static void main(String args[]) { …}

About main()

"main" is the function from which your program starts

Why public?

So that run time can call it from outside

Why static ?

it is made static so that we can call it without creating an object

What is String args[] ?

Way of specifying input at startup of application

Java—Why?

- Portable - Write Once, Run Anywhere
- Security has been well thought through
- Robust memory management
- Designed for network programming
- Multi-threaded (multiple simultaneous tasks)
- Dynamic & extensible (loads of libraries)
  - Classes stored in separate files
  - Loaded only when needed

Comments

```
/* This is a hello world example in Java
 * that will simply display Hello World
 *  on the monitor */
```

- Block Comment at start to describe purpose
  - /* … *comment* …*/
- Line comments used between statements
  - // *comment*

Class

```
public class HelloWorld {
   . . .
}
```

- At least one class per java file
  - Starts with keyword `public class`
  - Followed by class name
- All names have rules to follow
    - Each word in class name starts in uppercase (convention)
    - No punctuation (except underscore) and no spaces
    - Do not start with a number
    - Java file name must be same as class name

Main method

```
public class HelloWorld {
    public static void main(String[ ] args) {
        . . .
    }
}
```

- Java classes are structured into methods
  - Each java application must have one **`main`** method
- Main method always has same *signature*
  - Other methods differ

Statements

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

- Statements are terminated by semicolon
- Statements consist of construct and expression
  - Construct is the command
  - Expression is the data to be enacted upon

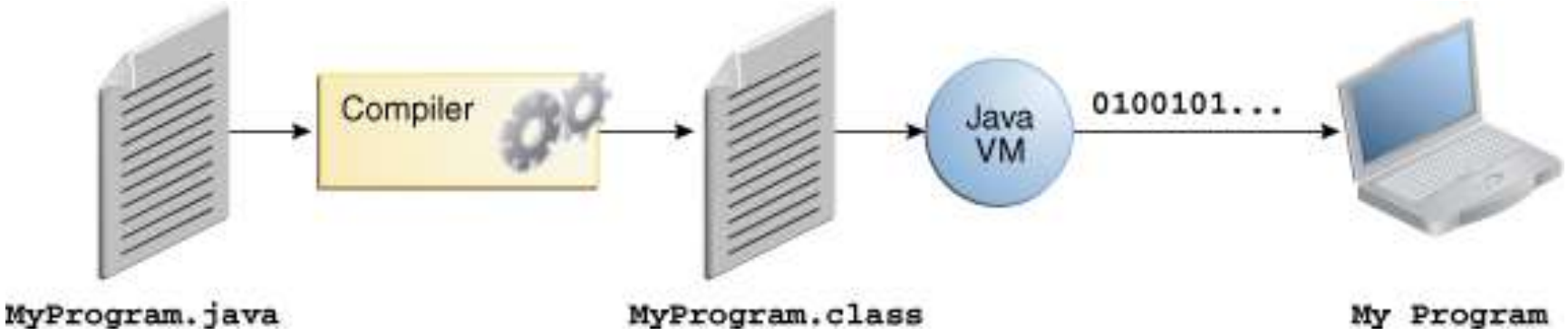
Compiling and Executing Java Programs

- Compilation

javac classname.java

•Execution

java classname



The Java Virtual Machine (JVM)

- Run-time Environment for Java programs.
- The JVM is machine dependent.
- The .class files contain Java bytecodes.
- Provides platform independence: Any platform having a JVM can execute the class files.
- The class files have a defined format that is followed by the Java compilers.
- Just In Time (JIT) compilation tries to increase speed.

Primitives Vs. Objects

Java Primitive Types

- Pre-defined by Java Programming Language and named by its reserved keyword.
- This means that you don't use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, 2;
double gamma = 1.2;
boolean valueOk = false;
```

- Homework: Find the range of values for all these primitive types.

Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
boolean	1 bit

Primitives Vs. Objects

Everything in Java is an “Object”, as every class by default inherits from class “Object” , except a few primitive data types, which are there for efficiency reasons.

Primitive Data Types

8 Primitive Data types of java

boolean, byte	→	1 byte
char, short	→	2 bytes
int, float	→	4 bytes
long, double	→	8 bytes

Primitive data types are generally used for local variables, parameters and instance variables (properties of an object)

Primitive datatypes are located on the stack and we can only access their **value**, while objects are located on heap and we have a **reference** to these objects

Also primitive data types are always passed by value while objects are always passed by reference in java. There is no C++ like methods

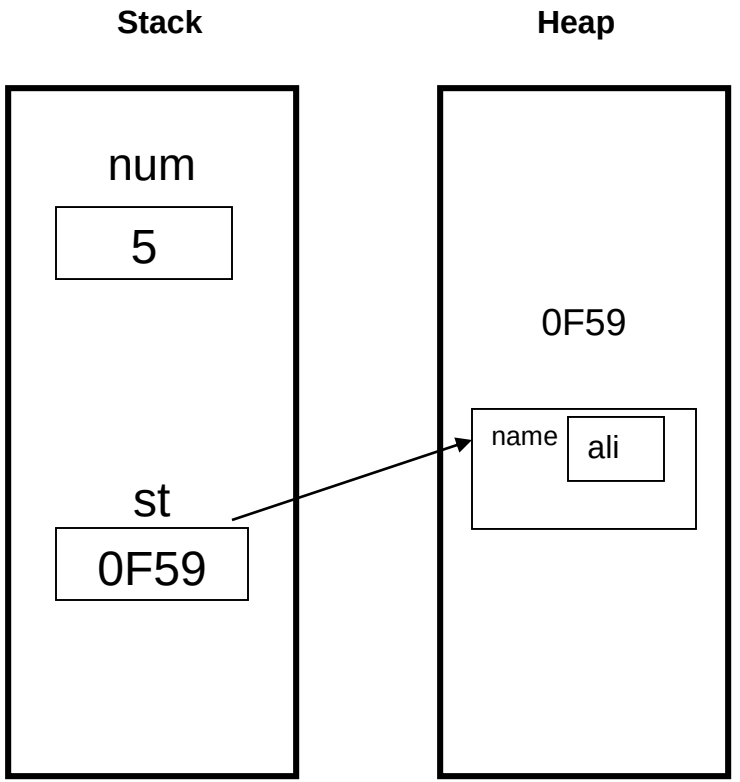
void someMethod(int &a, int & b) // not **available** in java

Stack vs. Heap

```
public static void main(String args[])
{
    int num= 5;

    Student  st = new Student();
                st.name = ali;

}
```



Primitives (cont)

For all built-in primitive data types java uses lowercase. E.g int , float etc

Primitives can be stored in arrays

You cannot get a reference to a primitive //as in c++ call by reference

To do that you need an Object or a Wrapper class

Wrapper Classes

Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

Wrapper Classes

Each primitive data type has a corresponding(اسی سے متعلق) object (wrapper class)

These Wrapper classes provides **additional functionality** (conversion, size checking etc), which a primitive data type can not provide

Primitive Data Type	Corresponding Object Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Wrapper Use

You can create an object of Wrapper class using a String or a primitive data type

```
Integer num = new Integer(4);   or
Integer num = new Integer("4");
```

Num is an object over here not a primitive data type

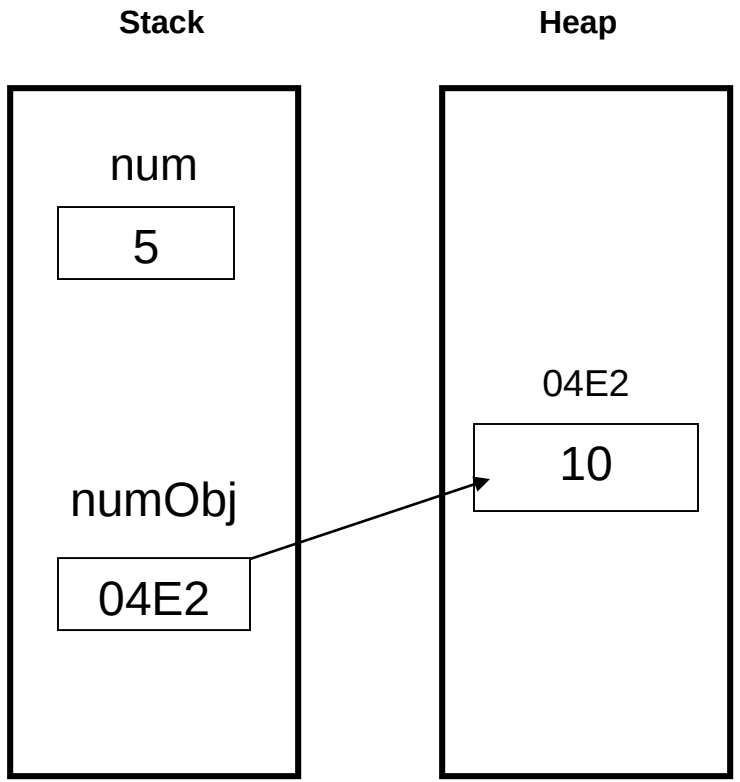
You can get a primitive data type from a Wrapper using the corresponding value function

```
int primNum = num.intValue();
```

Stack vs. Heap

```
public static void main(String args[])
{
    int num= 5;

    Integer numObj = new Integer (10);
}
```



Wrapper Uses

Defines useful constants for each data type

For example,

Integer.MAX_VALUE

Convert between data types

Use parse*Xxx* method to convert a String to the corresponding primitive data type

```
String value = “532";
  int d = Integer.parseInt(value);

String value = "3.14e6";
  double d = Double.parseDouble(value);
```

Wrappers: Converting Strings

Data Type	Convert String using either …
byte	`Byte.parseByte(`*string*`)`
	`new Byte(`*string*`).byteValue()`
short	`Short.parseShort(`*string*`)`
	`new Short(`*string*`).shortValue()`
int	`Integer.parseInt` [redacted]
	`new Integer(`*string*`).intValue()`
long	`Long.parseLong(`*string*`)`
	`new Long(`*string*`).longValue()`
float	`Float.parseFloat(`*string*`)`
	`new Float(`*string*`).floatValue()`
double	`Double.parseDouble(`*string*`)`
	`new Double(`*string*`).doubleValue()`

Input / Output

ROUGH

In **Java System Class**, we have 3 different types of field and 28 different types of method. Java System Class consists of following fields:-

SN	Modifier and Type	Field	Description
1	static PrintStrean	err	The "standard" error output stream.
2	static InputStream	in	The "standard" input stream.
3	static PrintStream	out	The "standard" output stream.

Console based Output

System.out

System class
- Out represents the screen

System.out.println()
- Prints the string followed by an end of line
- Forces a flush

System.out.print()
- Does not print the end of line
- Does not force a flush

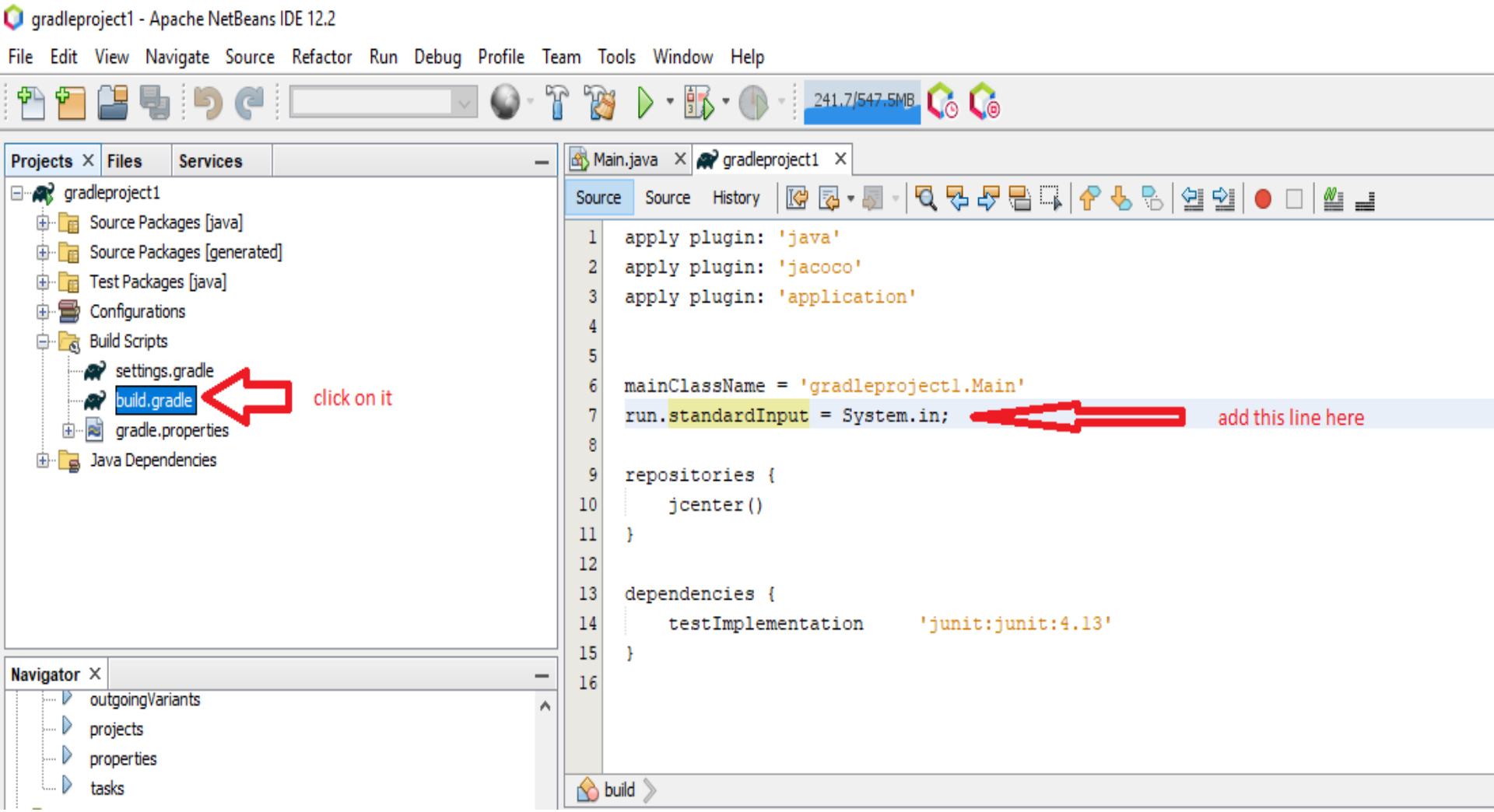
System.out.flush()
- Force a flush

Input / Output

```
/*  This program will takes the input (number) through GUI and prints its square on
  the console as well as on the GUI.  */
public class InputOutputTest {
   public static void main(String[] args) {
Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
}
}
```

Rough…..in case input is not working



Basic Mathematical Operators

- `* / % +` – are the mathematical operators
- `* / %` have a higher precedence than `+` or `–`

```
double val = a + b % d – c * d / b;
```

- Is the same as:

```
Double val = (a + (b % d)) –
                    ((c * d) / b);
```

Assignment Operators

- `=` Assignment operator
- When a calculation involves one variable on both sides we can use an assignment operator

  `+=` `-=` `*=` `/=` `%=`
- For example if we wish to increase the variable `num` by `10` the full calculation is

```
num = num + 10;
```

- As only `num` is being used we can apply the `+=` assignment operator

```
num += 10;
```

Unary Operators

- If an `int` variable is to be increased by `1`, then we can apply the pre/post unary incremental operator
  - `++num` or `num++`
- If an `int` variable is to be decreased by `1`, then we can apply the pre/post unary decremental operator
  - `--num` or `num--`
- We use these operators as part of an statement
  - Pre operator increments/decrements at start of statement
  - Post operator increments/decrements at end of statement

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

```
x = 2;
```

- A block is a compound statement enclosed in curly brackets:

```
{
        x = 2; y = 3;
}
```

- Blocks may contain other blocks

Methods

- A method is a standalone block of code, which
  - Is only run when invoked (by its name)
  - Designed to achieve a set task
  - May accept data when being invoked, via parameter passing
  - May or may not return a result, i.e. return type
- So far we have only written code in the main method
  - But now we will write code in separate methods

Method Format and Examples

- **Format**

  [*modifier*] [`static`] *returnType* *methodName* (*parameters*) {
  //method code
  }

- No return type example, (no body and no parameters)

```
private void emptyMethod(){
}
```

- Return type example (body, parameter and return line)

```
private static int getPerimeter(int length){
  return 4 * length;
}
```

Using Methods

- A method can be invoked by any code within the same class
  - However the `main` method is always the starting point for the whole program
  - We will often invoke methods from `main`
  - In which case the methods should be marked `static`
- To invoke a method we simply call the name of the method and supply any needed arguments

```
emptyMethod();
```

- If a method returns a value then we can assign the method call to a variable:

```
perimeter = getPerimeter(length);
```

Control Flow Statements

- Normally control flows from top to bottom in a method. Control flow statements break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code.
- Decision-making statements (if-then, if-then-else, switch)
- Looping statements (for, while, do-while)
- Branching statements (break, continue, return)

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 5 ) x = 10;
```

- If the value of x is less than 5, make x equal to 10
- It could have been written:

```
if ( x < 5 )
x = 10;
```

- Or, alternatively:

```
if ( x < 5 ) { x = 10; }
```

Relational Operators

==	Equal
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
  System.out.print(“x was changed”);
}
else {
  System.out.print(“x is unchanged”);
}
```

Nested if … else

```
if ( CONDITION1 ) {
  if ( CONDITION2 ) {
   System.out.println("Condition1 and
  Condition2 both are true");
  }
  else {
   System.out.println("Condition1 is true
  and Condition2 is not");
  }
}
else
{
  System.out.println("Condition1 is not
  true");
}
```

else if

- Useful for choosing between alternatives:

```
if ( CONDITION1 ) {
  // execute code block #1
}
else if ( CONDITION2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed,
  execute code block #3
}
```

The switch Statement

```
switch ( n ) {
  case 1:
   // execute code block #1
   break;
  case 2:
   // execute code block #2
   break;
   default:
   // if all previous tests fail then
   //execute code block #4
   break;
}
```

The for loop

- Loop n times

```
for ( i = 0; i < n; n++ ) {
  // this code body will execute n times
  // from  0 to n-1
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {
  for ( i = 0; i < 20; i++ ){
   // this code body will execute 200 times
  }
}
```

while loops

```
n=0
while(n<10) {
  System.out.print( " The value of n is" + n);
  n++;
}
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

do {… } while loops

```
n=0
do {
    System.out.print( " The value of n is" + n);
  n++;
} while(n<10);
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

break

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

```
for ( int i = 0; i < n, i++ ) {
  if ( CONDITION1) {
   // statements here
   break;
  }
} // program jumps here after break
```

return

- Exits a method with or without a value.
- Discussed earlier in Methods

Further Reading

- Oracle's *Java Tutorial*
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html