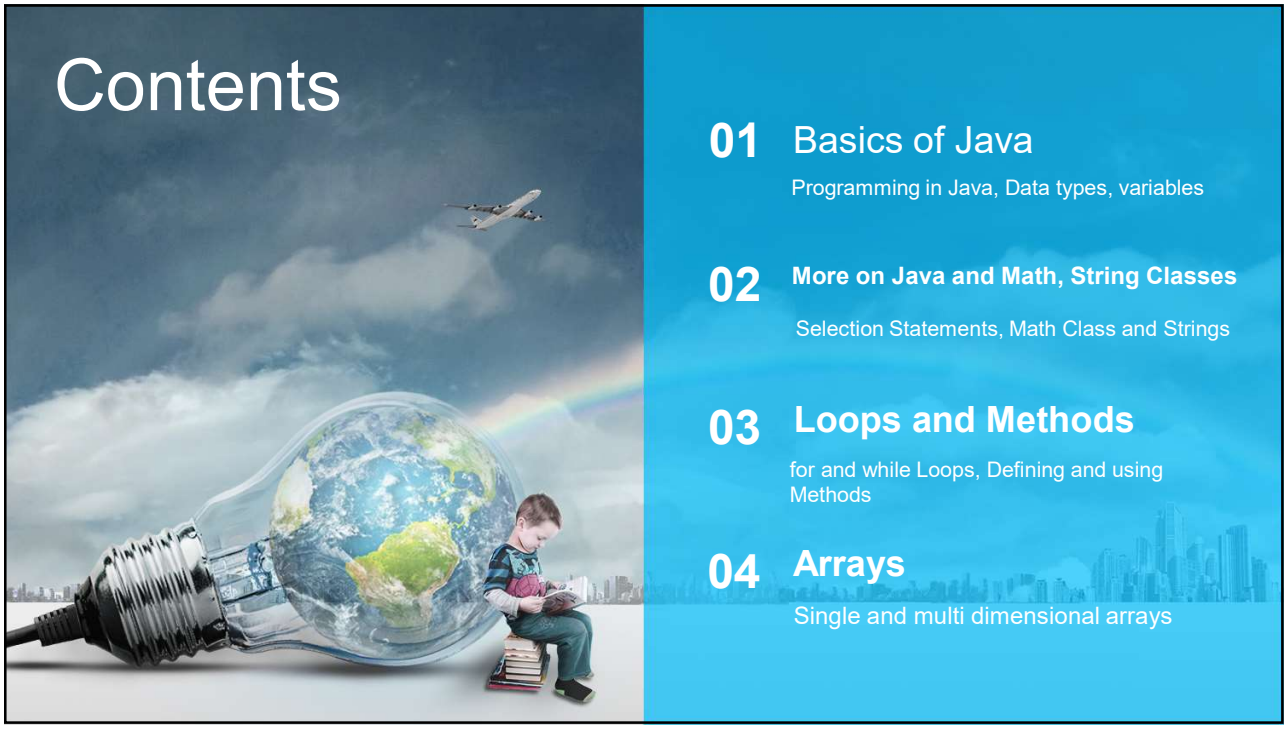
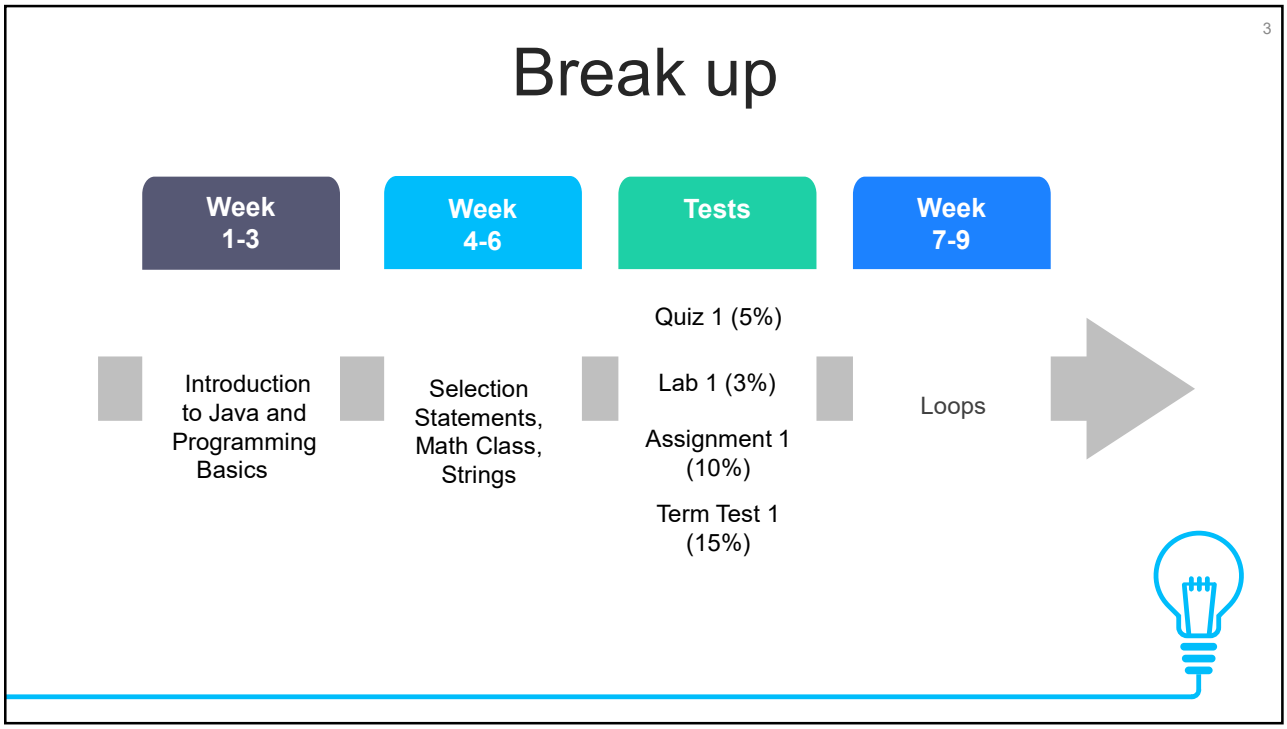

IN1044
Introduction to Programming
Instructor
Dr. Muhammad Waqar
muhammad.aslam@purescollege.ca

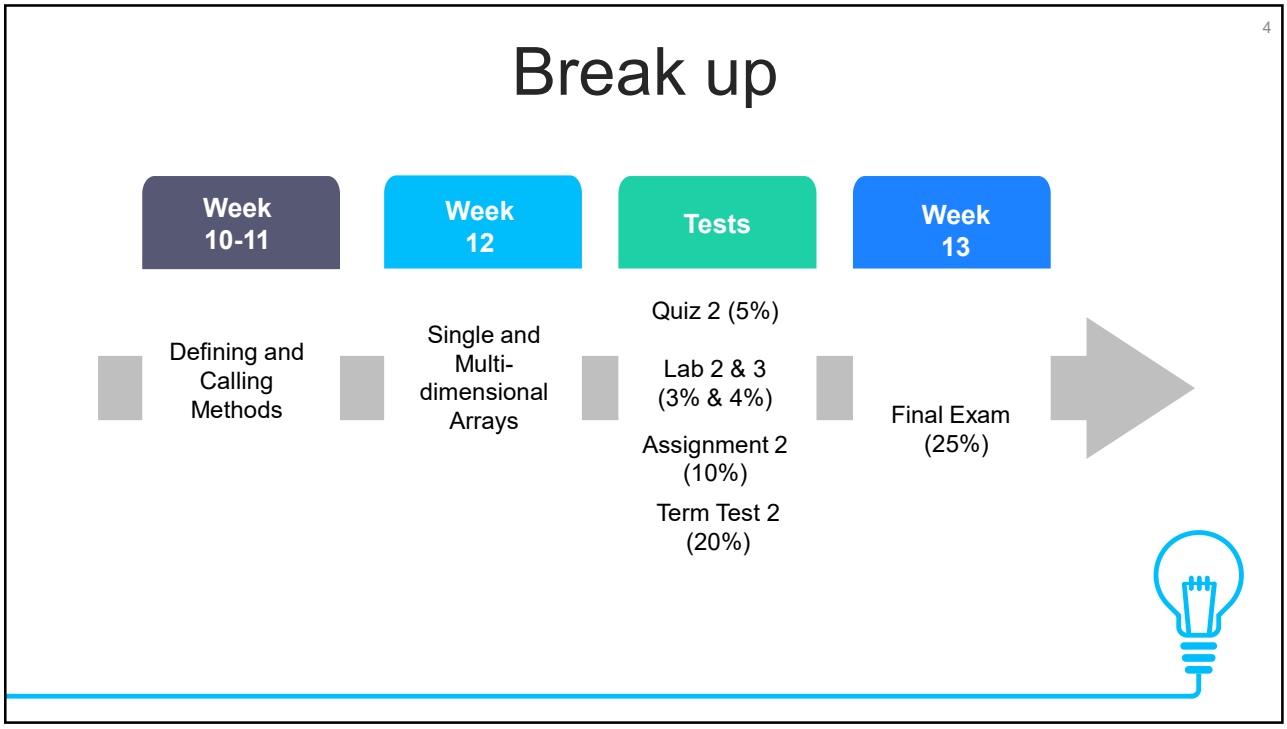
1


Contents
01 Basics of Java
Programming in Java, Data types, variables
02 More on Java and Math, String Classes
Selection Statements, Math Class and Strings
03 Loops and Methods
for and while Loops, Defining and using Methods
04 Arrays
Single and multi dimensional arrays

2


3
Break up
Week 1-3
Week 4-6
Tests
Week 7-9
Introduction to Java and Programming Basics
Selection Statements, Math Class, Strings
Quiz 1 (5%)
Lab 1 (3%)
Assignment 1 (10%)
Term Test 1 (15%)
Loops

3


4
Break up
Week 10-11
Week 12
Tests
Week 13
Defining and Calling Methods
Single and Multi-dimensional Arrays
Quiz 2 (5%)
Lab 2 & 3 (3% & 4%)
Assignment 2 (10%)
Term Test 2 (20%)
Final Exam (25%)

4

Grading Scheme

Test	Percentage
Assignments (2)	20% (10% each)
Quizzes (2)	10% (5% each)
Programming Labs (3)	10% (3%, 3%, 4%)
Term Test 1	15%
Term Test 2	20%
Final Exam	25%
Total	100%

5

Grading

Marks	Point	Meaning
A (80-100%)	4	EXCELLENT: Outstanding performance
B (70-79%)	3	GOOD: Better than average achievement
C (60-69%)	2	SATISFACTORY: Achievement sufficient to enable the student to progress in the course
D (50-59%)	1	MARGINAL: Minimum acceptable level of achievement; Exception: for courses where the passing grade is a “C”, a grade of “D” represents an insufficient achievement; must repeat.
F (0-49%)	0	UNSATISFACTORY: Insufficient achievement; must repeat. Student must consult with Program Coordinator.

The passing percentage for this course is 60% (Grade C, Point 2)

6

7

Textbooks

1. Introduction to Java Programming, Brief Version, Y. Daniel Liang
2. Java: The Complete Reference, Herbert Schildt

7

8

Google Class Room

- It will be used for sharing all the materials and any announcements related to this course
- Please see your google class room code at the end of lecture and use it to join google class room.
- Please use your college email address for joining class and also for any correspondence.

8

9

INTRODUCTION

CHAPTER 1

9

10

Computers and Languages

- Life in this age depends on smart devices
- Automobiles, Smart homes, Smart cities all of these depend on computers
- The computer program's role in this technology is essential; without a **list of instructions** to follow, the computer is virtually useless.
- Programming **languages** allow us to write those **programs** and thus make computers do something useful.

10

11

Elements of a Computer System

- The elements of a computer system fall into two major categories: hardware and software.
- ***Hardware*** is the equipment used to perform the necessary computations (central processing unit (CPU), monitor, keyboard, power supply, drives, RAM, etc.
- ***Software*** consists of the programs that enable us to solve problems with a computer (lists of instructions to perform).

11

12

Computer Hardware

Essentially, modern computers consist of:

1. Main memory
2. Secondary memory, which includes storage devices such as hard disks and flash drives
3. Central processing unit
4. Input devices, such as keyboards and mouse
5. Output devices, such as monitors and printers

12

13

Computer Software

Operating System

- The collection of computer programs that control the interaction of the user and the computer hardware is called the operating system (OS): e.g.: UNIX, Windows, MAC-OS X
- The operating system of a computer is the software that is responsible for directing all computer operations and managing all computer resources.

Application Software

- Application programs are developed to assist a computer user in accomplishing various tasks (i.e. word processing, financial calculations etc.) e.g. Notepad, MSWord, MATLAB etc.

13

14

Computer Languages

- Developing new software requires writing lists of instructions for a computer to execute.
- The language directly understood by a computer is called *machine language (or machine code)* and is a collection of binary numbers.
- For brevity Software developers rarely use the machine language directly because:
    1. It is not standardised – depends on the CPU
    2. Difficult to use – long development times

However it does have some advantages

1. Faster execution
2. Requires less memory.

14

15

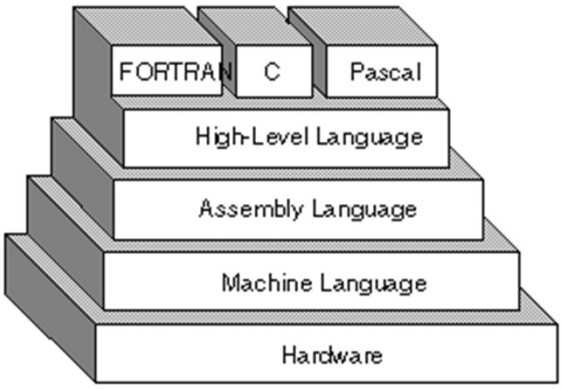
High-level Languages

- To write programs that are independent of the CPU on which they will be executed, software designers use high level languages, which combine algebraic expressions and symbols taken from English.
- There are many high-level languages available.
- Fortran (Scientific), Cobol (Business), LISP (Artificial Intelligence (AI)) Prolog (AI), Java (Internet), C++ (General), Python (General)

15

16

Programming Languages

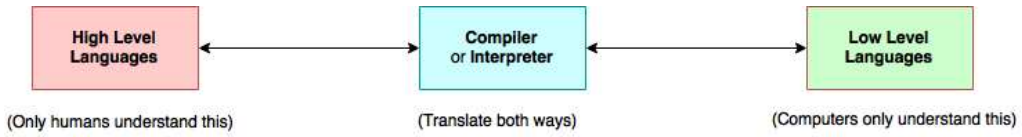


https://www.webopedia.com/TERM/H/high_level_language.html

16

17

Execution Steps



https://thebittheories.com/levels-of-programming-languages-b6a38a68c0f2

17

18

Software Development Method

1. Specify the problem requirements.
2. Analyze the problem.
3. Design the algorithm to solve the problem.
4. Implement the algorithm.
5. Test and verify the completed program.
6. Maintain and update the program.

18

19

Problem Specification

Specifying the problem requirements allows you to:

1. State the problem clearly and unambiguously
2. Gain a clear understanding of what is required for its solution
3. Eliminate unimportant aspects

To achieve this goal you may find you need more information from the person who posed the problem.

19

20

Analysis

Analyzing the problem involves identifying the:

1. *Inputs* - the data you have to work with
2. *Outputs* - the desired results
3. Additional requirements or constraints on the solution.

20

21

Design

Designing the algorithm

1. Develop a list of steps called an algorithm to solve the problem
2. Verify that the algorithm solves the problem as intended. Writing the algorithm is often the most difficult part of the problem-solving process. Coding and algorithm is simple once you know the programming language syntax

Don't attempt to solve every detail of the problem at the beginning.

Top-down Design Approach

1. List the major steps, or subproblems, that need to be solved
2. Solve the original problem by solving each of its sub-problems.

21

22

Implementation

Implementing the algorithm

- Involves writing the algorithm as a computer program.
- You must convert each step of the algorithm into one or more statements in a programming language.
- Requires a full grip on programming language syntax and constraints

22

23

Testing and Verification

Testing and verification

- Requires testing the completed program to verify that it works as desired.
- Don't rely on just one test case.
- Run the program several times using different sets of data, making sure that it works correctly for every situation provided for in the algorithm.

23

24

Maintenance

Maintaining and updating

- Involves modifying a program to remove previously undetected errors and to keep it upto-date as government regulations or company policies change.
- Many organizations maintain a program for five years or more, often after the programmers who originally coded it have left or moved on to other positions.

24

25

Development of C Language

- The creation of C language changed the way programming was approached and thought about
- The creation of C was direct result of need of a structured, efficient high level language that could replace assembly language
- C was first developed in 1970s by Dennis Ritchie
- C was formally standardized in 1989
- C++ is enhancement of C which supports Objected oriented Programming (OOP) that helps organize large complex problems.

25

26

Development of Java

- Language innovation is driven by two factors: improvement in art of programming and computing environment
- Java inherits the rich legacy of C and C++ and adds new features required by current state of art systems
- Java offers features that streamline programming for a distributed architecture
- Java was initially conceived in 1991 by a group of programmers at Sun Microsystems and was initially called “Oak” but it was renamed as Java in 1995.

26

27

Purpose of Java

- C++ required a different compiler for each of these devices as these had a different kind of CPU and compilers are expensive and time consuming
- The primary motivation was the need to have a platform independent language that could be used in consumer electronics, microwave, toasters remote control etc.
- The need for platform independent language increased after the development of internet as internet connected devices consisted of various types of computers, CPUs and operating systems.
- Internet caused Java programmers to switch from consumer electronics to internet

27

28

Java and C/C++

- Java is directly related to C/C++ as it inherits syntax from C/C++.
- Its object oriented model is adapted from C++
- Any programmer who knows C/C++ can easily understand Java and vice versa.
- Java takes the best of the past languages and adds new features needed for the online and platform independent environment
- Although java was influenced by C/C++, it is not an enhanced version of C/C++ and is not compatible with C/C++.
- Java is not designed to replace C/C++ and both of these will coexist as these sort different sorts of problems.

28

29

Java's Success

- Java's success lies in its adaptability to changing programming environments
- Java's release cycles is about 1.5 years compared to 10 years for other languages
- Java is more secure and portable

29

30

Java's Magic: The Bytecode

- Java's security and portability advantage lies in the fact that it java compiler does not produce an executable file, it produces java bytecode
- Bytecode is a highly optimized set of instructions designed to be executed by java runtime systems called java virtual machine (JVM).
- Translating a java program into bytecode makes it easier to run on a variety of platforms as only JVM needs to be updated
- Although details of JVM will differ from platform to platform, all understand same bytecode
- Since JVM is in control so it prevents from generating side effects and makes system more secure

30

31

Java's Magic: The Bytecode

- Compiling program to an intermediate form and then interpreting it makes process slower compared to if it was compiled to an executable code, the difference is not great because bytecode is highly optimized
- It is not practical to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time.

31

32

Java Buzz Words

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

32

33

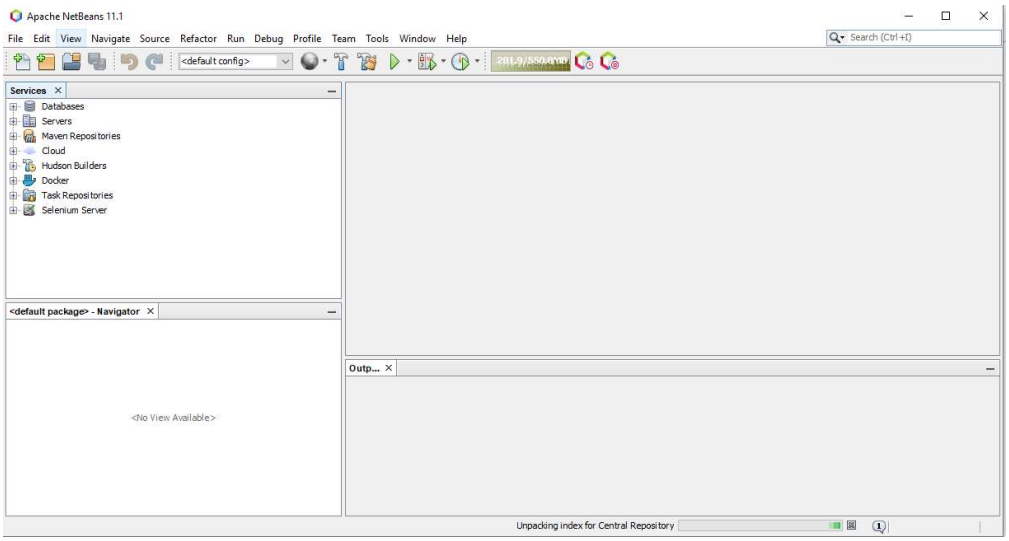
Running Java Code

- Different learning environments available for running Java program
- Popular are NetBean, Eclipse, BlueJ, Jedit, Jgrasp etc.
- In this course we will be using NetBeans IDE.
- Go to start menu and search for NetBeans and open it

33

34

Using NetBeans

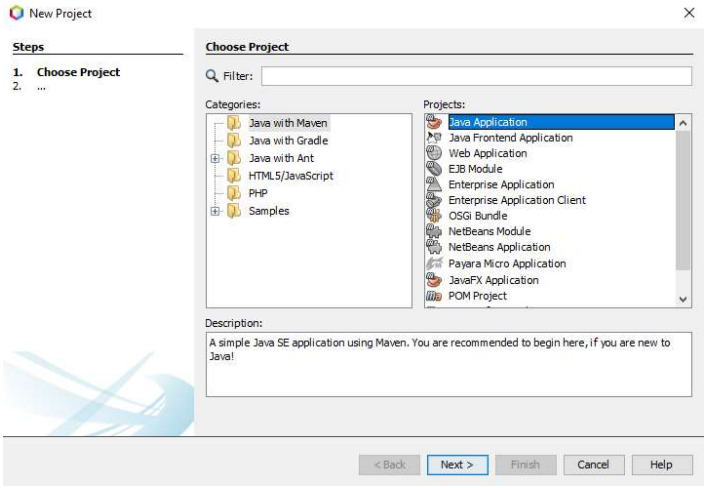


34

35

Creating New Project

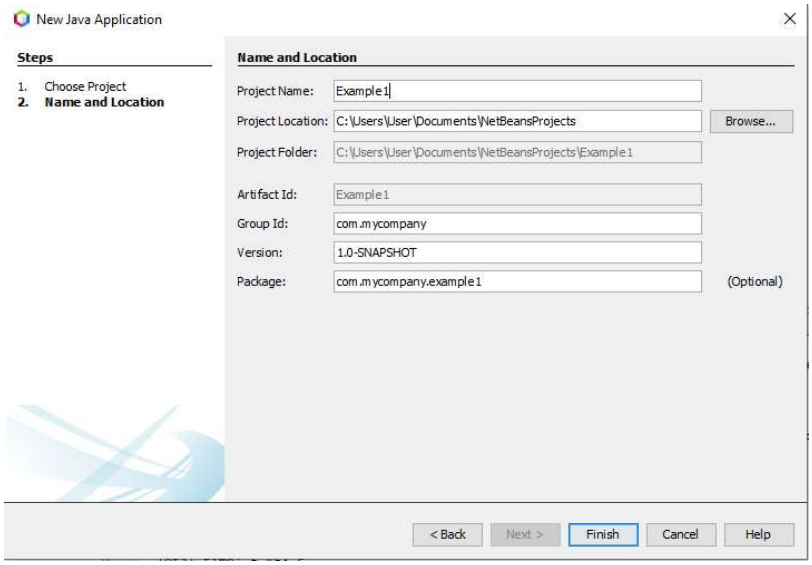
Go to File> New Project



35

36

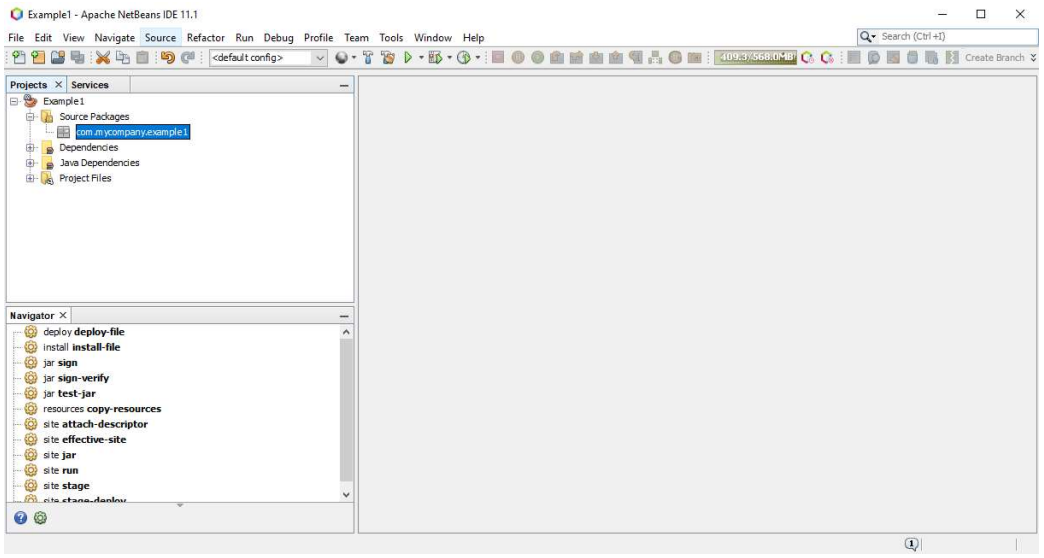
Creating New Project



36

37

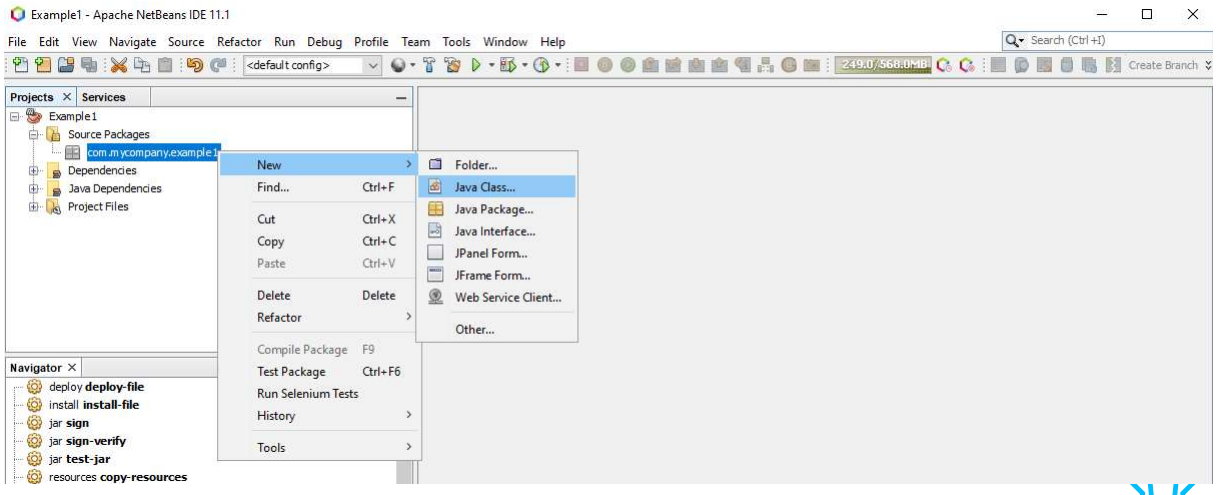
Creating New Project



37

38

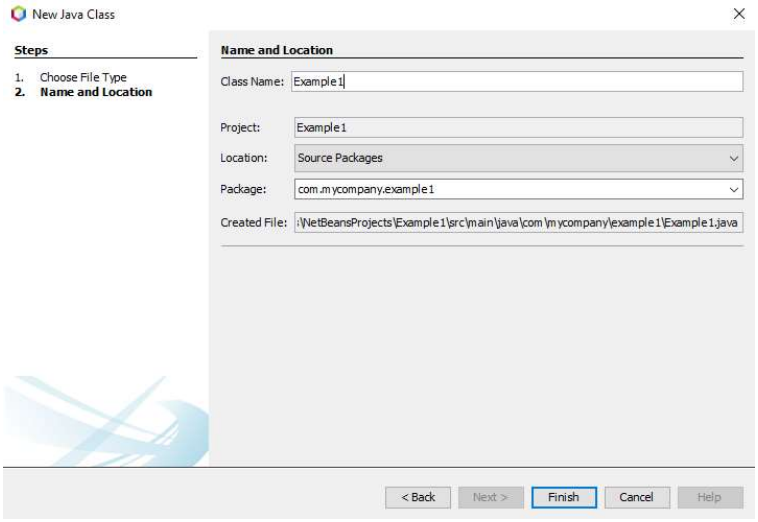
Creating New Project



38

39

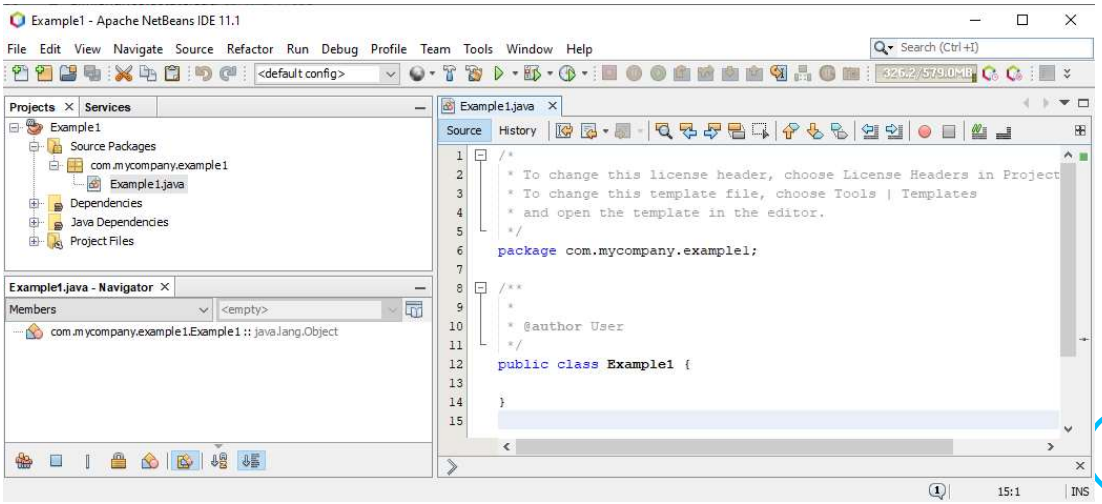
Creating New Project



39

40

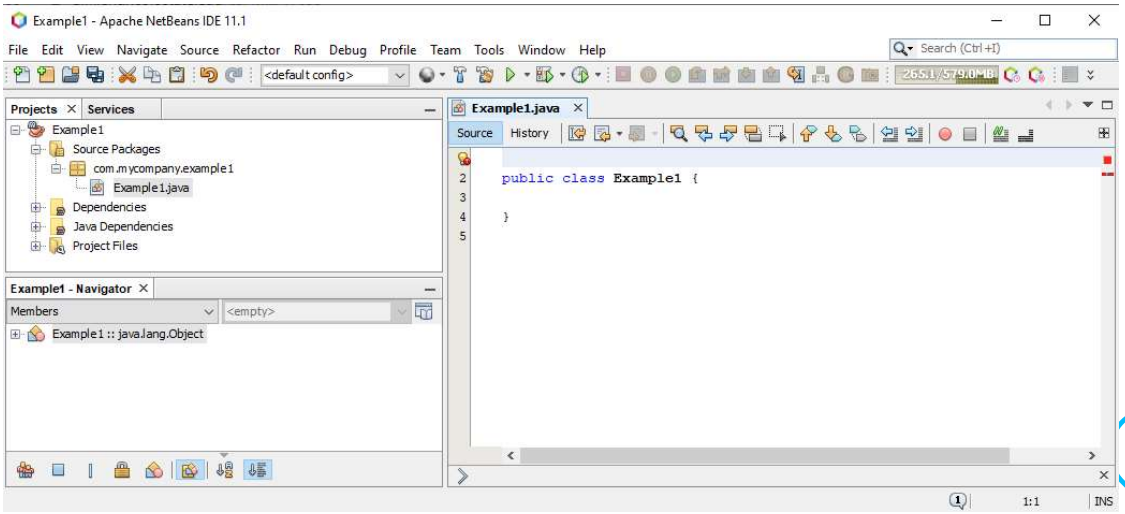
Creating New Project



40

41

Creating New Project



41

42

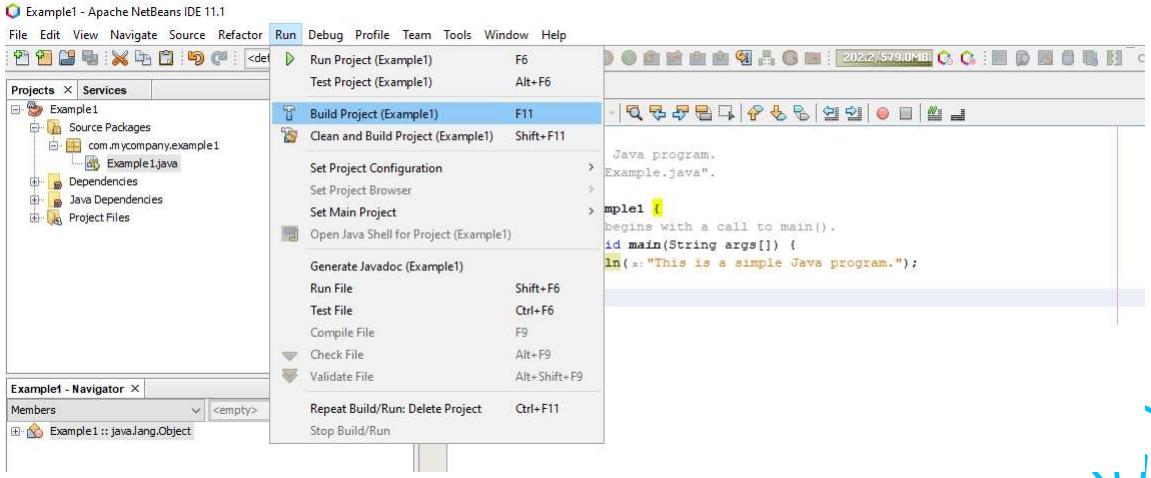
Writing first Program

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
public class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
}
}
```

42

43

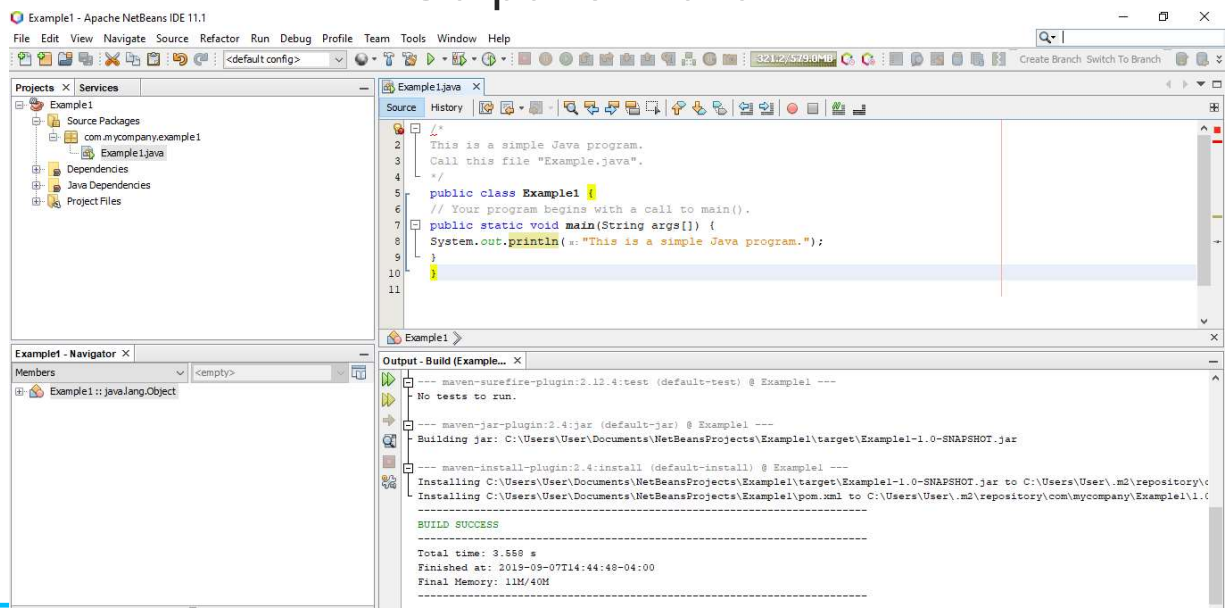
Building first Program



43

44

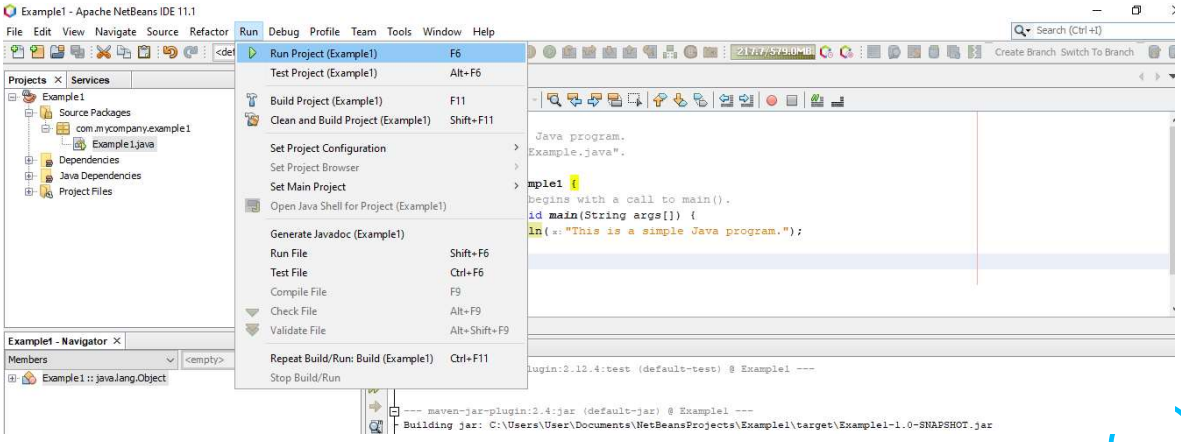
Output of Build



44

45

Running Program



45

46

Output of Run

46

47

A closer look at the program (Comments)

```
/*
This is a simple Java program.
Call this file "Example.java".
*/

// Your program begins with a call to main().
```

The contents of a comment are ignored by the compiler. Instead, a comment describes or explains the operation of the program to anyone who is reading its source code.

47

48

public key word

```
public class Example1 {
}
```

The public keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.

The opposite of public is private, which prevents a member from being used by code defined outside of its class

48

49

static and void

public static void main(String args[ ])

The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class.

The keyword **void** simply tells the compiler that **main( )** does not return a value.

49

50

main() method

public static void main(String args[ ])

- All Java applications begin execution by calling main( )
- main( ) must be declared as public, since it must be called by code outside of its class when the program is started.
- Keep in mind that Java is case-sensitive. Thus, Main is different from main.
- It is important to understand that the Java compiler will compile classes that do not contain a main( ) method. But java has no way to run these classes.

50

51

String args[ ]

public static void main(String args[ ])

- String args[ ] declares a parameter named args, which is an array of instances of the class String.
- Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

51

52

Printing a statement

System.out.println("This is a simple Java program.");

This line outputs the string “This is a simple Java program.” followed by a new line on the screen.

Output is actually accomplished by the built-in **println( )** method. In this case, **println( )** displays the string which is passed to it.

println( ) can be used to display other types of information, too.

52

53

Another Simple Program

```
class Example2 {
public static void main(String args[]) {
int num;                          // this declares a variable called num
num = 100;                        // this assigns num the value 100
System.out.println("This is num: " + num);
num = num * 2;
System.out.print("The value of num * 2 is ");
System.out.println(num);
}
}
```

53

54

1

2

An Overview of Lecture 1

We discussed the following in lecture1

- Computer Hardware, software and languages
- Software development model
- A comparison of Java with other languages
- Advantages of Java
- Using NetBeans to compile java programs
- Discussed different parts of a simple program

2

3

A Simple Program

```
public class Example2 {

public static void main(String args[]) {
   System.out.println("John Smith");
}
}
```

Output of this program is

John Smith

3

4

Practice Problems

1- Write a program which prints your name vertically

2- Write a program that displays your name on two lines inside a rectangle of asterisks as follows:

```
**********
*  First name  *
*  Surname     *
**********
```

Make sure the asterisks are properly aligned. To do this you will need to insert spaces (blanks) in the right place.

4

5

Practice Problem 1 code

```
public class Example2 {
public static void main(String args[]) {
   System.out.println("J");
   System.out.println("o");
   System.out.println("h");
   System.out.println("n");
   System.out.println("");
   System.out.println("S");
   System.out.println("m");
   System.out.println("i");
   System.out.println("t");
   System.out.print("h");
}
}
```

5

6

Practice Problem 2 code

```
public class Example2 {

public static void main(String args[]) {
   System.out.println("****************");
   System.out.println("*  John       *");
   System.out.println("*  Smith      *");
   System.out.println("****************");
}
}
```

6

7

The Primitive Data Types

- Java defines eight primitive data types: byte, short, int, long, float, double, char and boolean
- Integers: This group includes which are for whole valued 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 true/false values

7

8

Integers

- Java has four integer types: byte, int, short and long
- All of these are signed, positive and negative values
- Java does not support positive only, unsigned 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

8

9

Byte and short

- The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127.
- Byte is useful when you're working with raw binary data that may not be directly compatible with Java's other built-in types.
- Byte variables are declared by use of the byte keyword.

```
byte b, c;
```

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

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

9

10

Integer

- The most commonly used integer type is **int**. It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647.
- In addition to other uses, variables of type **int** are commonly employed to control loops and to index arrays.
- When **byte** and **short** values are used in an expression they are *promoted* to **int** when the expression is evaluated. Therefore, **int** is often the best choice when an integer is needed.

```
int a, b;
```

10

11

long

- The **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. This makes it useful when big, whole numbers are needed.

11

12

Variables

- The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initializer.
- In addition, all variables have a scope, which defines their visibility, and a lifetime.
- In Java, all variables must be declared before they can be used.

12

13

Variables

type identifier = value, identifier = value ;

- The type is one of Java's atomic types, or the name of a class or interface.
- The identifier is the name of the variable.
- You can initialize the variable by specifying an equal sign and a value.
- To declare more than one variable of the specified type, use a comma separated list.

13

14

Variables Examples

```
int a, b, c; // declares three ints, a, b, and c.

a=5;    // assigns a value 5 to already declared int variable a (initialization)

int d = 3, e, f = 5; // declares three more ints, initializing d and f.

byte z = 22; // initializes z.

double pi = 3.14159; // declares an approximation of pi.

int g = d*f;              // dynamic initialization
```

14

15

Simple program using Integers

- This program calculates square of a number

```
public class Example2 {

public static void main(String args[]) {

int number = 5;
   System.out.println("Square of number is: " + (number*number));

}
}
```

15

16

Another simple program using Integers

- Write a program which converts 3 hours and 25 minutes into seconds.

Steps

i- Define four int variables '***hours***', '***minutes***', '***total_minutes***', '***total_seconds***'

1- Assign value 3 to variable '***hours***' for saving hours value

2- Assign value 25 to variable '***minutes***' for saving minutes

3- Calculate value of '***total_minutes***' by converting '***hours***' into minutes and by adding '***minutes***' to it.

4- Calculate value of '***total_seconds***' by multiplying '***total_minutes***' to 60

5- Print '***total_seconds***'

16

17

Program Code

```
public class Example {
public static void main(String args[]) {
int hours, minutes, total_minutes, total_seconds;
   hours = 3;
   minutes = 25;
   total_minutes = hours * 60 + minutes;
   total_seconds = total_minutes * 60;
   System.out.println("Total Seconds are:  " + total_seconds + "
seconds");
}
}
```

17

18

Floating-Point Types

- Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.
- Calculations such as square root, sine and cosine, result in a value whose precision requires a floating-point type.

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 high_temp = 33.9f; // Always append f declaring float
- double pi = 3.14;

18

19

A simple program

Write a program which converts 25 miles into kilometers.

Steps:

1- Define a double variable '***miles***' which has value equal to 25. (stores miles)
2- Dynamically initialize another double variable '***Kilo_meters***' by converting 25 miles into kilometers. (Hint: 1 mile = 1.609 km)
3- Print value of ***Kilo_meters***

19

20

Code

```
public class conversion {

public static void main(String args[]) {

 double miles = 25;
 double kilo_meters = miles * 1.609;
    System.out.println(miles + " miles =  " + kilo_meters + " kilometers");

}
}
```

20

21

Another simple program

Write a program which calculates area of a circle with radius = 10.8

Steps:

1- initialize variable '***radius***' with radius value
2- initialize variable '***pi***' with 3.1416
3- dynamically initialize '***area'*** using '***radius***' and '***pi***' (Hint: ***area*** = ***pi**r*^{2})

21

22

Code

```
// Compute the area of a circle.

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

22

23

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.
- It is a unification of dozens of character sets, such as Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul, and many more. For this purpose, it requires 16 bits.
- Thus, in Java char is a 16-bit type. The range of a char is 0 to 65,535 and there are no negative chars.
- Char value is declared in single quotes in order to be taken as character

23

24

Sample program using char

Here is a program that demonstrates char variables:

```
// 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);
}
}
```

24

25

Exercise Program

In previous program you printed ‘X’ by using its Unicode. Now try to get this out put i.e.

ch1 and ch2: X x

But this time X must be saved in ch1 as character using single quotes and x (lowercase) must be saved in ch2 using its Unicode.

(Hint: Unicode of a is 97 so you can predict Unicode of x)

25

26

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, as in the case of a < b.
- boolean is also the type required by the conditional expressions that govern the control statements such as if and for.
- The true literal in Java does not equal 1, nor does the false literal equal 0.
- Below are examples of boolean variables.

boolean a = true;

boolean b = false;

26

27

Sample Program

```
Public class booltest{
public static void main(String args[]) {
int number1 = 10;
int number2 = 20;
boolean check = number1>number2;

System.out.println("The statement that number1 is greater than
number2 is " + check);
}
```

Now try changing statement boolean check = number2>number1 and see what result you get.

27

28

Strings

- String literals in Java are specified like they are in most other languages—by enclosing a sequence of characters between a pair of double quotes.
- Examples of string literals are

```
String String1 = “Hello World”;
System.out.println(String1);
```

The above example prints “Hello World” stored in string1.

28

29

Escape sequences

- Escape sequences are used for special character insertion or for special action

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal Unicode character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

29

30

Examples of Escape Sequences

- If you want to print “Hello World”, you need escape sequence

- System.out.println("Hello world"); // just prints Hello World
- System.out.println(""Hello world""); // gives error as text is out of “”
- System.out.println("\"Hello world\""); // Correct

- System.out.println(“This is backslash \"); // error
- System.out.println(“This is backslash \\");

- \r and \n behave same in most windows platforms i.e. any text after these will move to next line (\r used for new line in MAC OS)

30

31

Practice programs

- Write a program that displays your name vertically. Use print() statement only once. (Hint: use escape sequence)
- Write a program which produces the following output and use \t escape sequence to give tab

  My name is: Bob Smith

31

32

The scope of variables

- Java allows variables to be declared within any block.
- A block begins with an opening curly brace and ended by a closing curly brace. A block defines a scope.
- A variable defined inside a scope is only known inside that scope.
- A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.
- Many other computer languages define two general categories of scopes: global and local. However, these traditional scopes do not fit well with Java's strict, object-oriented model.

32

33

Example of variables scope

```
class Scope {
public static void main(String args[]) {
int x; // known to all code within main
x = 10;
if(x == 10) { // start new scope
int y = 20; // known only to this block
// x and y both known here.
System.out.println("x and y: " + x + " " + y);
x = y * 2;
}
// y = 100; // Error! y not known here
// x is still known here.
System.out.println("x is " + x);
}
}
```

33

34

Type Conversions and Casting

- It is fairly common to assign a value of one type to a variable of another type.
- If the two types are compatible, then Java will perform the conversion automatically. For example, it is always possible to assign an int value to a long variable.
- Not all types are compatible, and thus, not all type conversions are implicitly allowed. For instance, there is no automatic conversion defined from double to byte.
- It is still possible to obtain a conversion between incompatible types. To do so, you must use a cast, which performs an explicit conversion between incompatible types

34

35

Java's Automatic Conversions

- An automatic type conversion will take place if the following two conditions are met:

1- The two types are compatible.

2- The destination type is larger than the source type.

- When these two conditions are met, a ***widening*** conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required.
- For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other.
- However, there are no automatic conversions from the numeric types to boolean.

35

36

Java's Automatic Conversions

- byte to short, int, long, float, or double
- short to int, long, float, or double
- char to int, long, float, or double
- int to long, float, or double
- long to float or double
- float to double

36

37

Java's Automatic Conversion Examples

```
int integer1 = 23;
float float1 = 23.45f, float2;
long long1;
double double1;
float2 = integer1;      // same width integer to float, correct
long1 = integer1;       //  integer to bigger width integer, correct
float1 = long1;         //  long to float also allowed
double1 = integer1;     // integer to bigger width float, correct
double1 = float1;       // float to double, correct
// integer1 = long1;    // bigger integer to smaller not allowed
//long1 = float1;       // not allowed float to integer
// float1 = double1;    // not allowed
```

37

38

Casting Incompatible types

- If you want to assign an int value to a byte variable, this conversion will not be performed automatically, because a byte is smaller than an int. You will have to do ***casting*** for this
- This kind of conversion is sometimes called ***narrowing conversion***, since you are explicitly making the value narrower so that it will fit into the target type.
- For example, the following fragment casts an int to a byte. If the integer's value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range.
- int a; byte b;
- b = (byte) a;

38

39

Example of Casting

```
public class int_to_byte {

public static void main(String args[]) {
int integer1 = 100;
byte byte1 = (byte) integer1;
System.out.println("Value of integer and byte is " + integer1 + " " +
byte1);
}
}
```

What is the maximum value of int that could be casted to byte?

39

40

Truncation

- A different type of conversion will occur when a floating-point value is assigned to an integer type: truncation.
- Since integers do not have fractional components, when a floating-point value is assigned to an integer type, the fractional component is lost.
- For example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1.The 0.23 will have been truncated.
- If the size of the whole number component is too large to fit into the target integer type, then that value will be reduced modulo the target type's range.

40

41

Truncation example

```
class Conversion {
public static void main(String args[]) {
byte b;
int i = 257;
double d = 323.142;
System.out.println("\nConversion of int to byte.");
b = (byte) i;
System.out.println("i and b " + i + " " + b);
System.out.println("\nConversion of double to int.");
i = (int) d;
System.out.println("d and i " + d + " " + i);
System.out.println("\nConversion of double to byte.");
b = (byte) d;
System.out.println("d and b " + d + " " + b);
}
}
```

41

42

Automatic Type Promotions

- In certain expression, Java automatically promotes types of variables during operation. For example

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

- The result of the intermediate term a * b easily exceeds the range of either of its byte operands.
- To handle this kind of problem, Java automatically promotes each byte, short, or char operand to int when evaluating an expression. This means that the subexpression a * b is performed using integers—not bytes. Thus, 2,000, the result of the intermediate expression, 50 * 40, is legal even though a and b are both specified as type byte.

42

43

Automatic Type Promotions

- Automatic promotions can sometimes cause confusing compile-time errors. For example, this seemingly correct code causes a problem:

```
byte b = 50;
b = b * 2;          // Error! Cannot assign an int to a byte!
```

- In this case when the expression was evaluated, the result has also been promoted to int. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.
- This is true even if, as in this particular case, the value being assigned would still fit in the target type.
- If you still want to save result into a byte, casting would be required.

44

The Type Promotion Rules

- All byte, short, and char values are promoted to int.
- Then, if one operand is a long, the whole expression is promoted to long.
- If one operand is a float, the entire expression is promoted to float. If any of the operands is double, the result is double.
- An example on next page gives detailed explanation of automatic promotions.

Automatic Promotion Example

```
class Promote {
public static void main(String args[]) {
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
float f = 5.67f;
double d = .1234;
double result = (f * b) + (i / c) - (d * s);
System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
System.out.println("result = " + result);
}
}
```

In the first subexpression, f * b, b is promoted to a float and the result of the subexpression is float.

Next, in the subexpression i / c, c is promoted to int, and the result is of type int.

Then, in d * s, the value of s is promoted to double, and the type of the subexpression is double.

45

46

1

2

An Overview of Lecture 2

We discussed the following in lecture2

- Primitive Data types, byte, short, int, long, float, double, char, boolean, String
- Declaration of variables, variable scope
- Escape Sequences
- Type Promotion and Casting

2

3

Getting Input from User

- We can ask the user to specify the value of variable during execution rather than initializing it ourselves during writing a program.
- Different kind of packages are available for getting input from user including Bufferreader class, Scanner class and Console class.
- Whichever class you use you will have to first import its package and then will have to instantiate an object to use methods of that class.
- An example is given on next page which used Scanner class to get input from user and then displays input using println.

3

4

Program for Getting Input from User

```
import java.util.Scanner;

public class GetInputFromUser {
public static void main(String args[]) {

Scanner input = new Scanner(System.in);
System.out.print("Please eneter an integer: ");
int number1 = input.nextInt();
number1 = number1*2;
System.out.println("Twice of integer is " + number1);
System.out.print("Please enter a float: ");
float number2 = input.nextFloat();
number2 = number2*4;
System.out.println("Four times of your float is "+number2);
}
}
```

4

5

Practice Program

Write a program which asks user to enter radius. Then it calculates the area of circle using radius and prints the area on screen. The output of program should be as shown below.

```
Please enter radius of circle: 4
Area of circle is 50.2656
```

5

6

Practice Program Code

```
public class Example1 {

public static void main(String args[]) {

Scanner input = new Scanner(System.in);
final double PI = 3.1416;   // final is used for constant
System.out.print("Please enter radius of circle: ");
double radius = input.nextDouble();
double area = PI*radius*radius;
System.out.println("Area of circle is "+area);
}
}
```

6

7

Operators

- Operators are used to apply some operation on variables and values.
- The value or variables are called operands, while the operation (to be performed between the two operands) is defined by an operator
- Java provides a rich operator environment.
- Most of its operators can be divided into the following four groups: *arithmetic, bitwise, relational, and logical*.
- Java also defines some additional operators that handle certain special situations.

7

8

Arithmetic Operators

- The operands of the arithmetic operators must be of a numeric type. You cannot use them on boolean types, but you can use them on char types, since the char type in Java is, essentially, a subset of int.

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

8

9

A simple program using arithmetic operators

```
class Example2 {
public static void main(String args[]) {
// arithmetic using integers
System.out.println("Integer Arithmetic");
int a = 1 + 1;
int b = a * 3;
int c = b / 4;
int d = c - a;
int e = c- d;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
System.out.println("e = " + e);
}
}
```

9

10

Simple Arithmetic Problems

What is the value in i after executing the following program segments?

1-
```
int i = 12;
int j = 5;
i = i / j;
```

2-
```
int i = 12;
int j = 5;
i = i % j;
```

3-
```
int i = 10;
int j = 5;
i = i / j;
```

4-
```
int i = 12;
int j = 20;
int k = 5;
i = i * 6 + j / 2 - k;
```

10

11

Operators Precedence

Precedence	Operator
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Relational)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

11

12

Arithmetic Overflow

What is the final value stored in *i* in this program? Is this what you expected?

```
class Test {
  // A program to demonstrate integer overflow

  public static void main(String[] args) {
    int i = 2000000000;
    int j = 2000000000;
    i = i + j;
    System.out.println("i + j = " + i);
  }
}
```

12

13

Modulus Operator

- The modulus operator, %, returns the remainder of a division operation.
- It can be applied to floating-point types as well as integer types.

```
// Demonstrate the % operator.
class Modulus {
public static void main(String args[]) {
int x = 42;
double y = 42.25;
System.out.println("x mod 10 = " + (x % 10));
System.out.println("y mod 10 = " + (y % 10));
}
}
```

13

14

Arithmetic Compound Assignment Operators

- Java provides special operators that can be used to combine an arithmetic operation with an assignment. Statements like the following are quite common in programming:

- a = a + 4;
- In Java, you can rewrite this statement as shown here:
- a += 4;
- This version uses the += compound assignment operator. Both statements perform the same action: they increase the value of a by 4.
- Compound assignment can be applied to any operator e.g. a *= 4 (is same as a= a*4;) a/=4 (a= a/4), a%=4 (a=a%4) etc.
- The compound assignment operators provide two benefits. First, they save you a bit of typing, because they are “shorthand” for their equivalent long forms. Second, they are implemented more efficiently by the Java run-time system than are their equivalent long forms.

14

15

Arithmetic Compound Assignment Operators

What will be the values in a, b and c after executing these commands?

```
int a = 1;
int b = 2;
int c = 3;
a += 5;
b *= 4;
c += a * b;
c %= 6;
```

15

16

Increment and Decrement Operator

- The ++ and the −− are Java's increment and decrement operators.
- The increment operator increases its operand by one. The decrement operator decreases its operand by one. For example, this statement:
- x = x + 1;

can be rewritten like this by use of the increment operator:

- x++; // is same as x = x+1;

Similarly, this statement:

- x - - // is same as x = x-1;

16

17

Increment and Decrement Operator

- These operators are unique in that they can appear both in postfix form, where they follow the operand as just shown, and prefix form, where they precede the operand.
- In the prefix form, the operand is incremented or decremented before the value is obtained for use in the expression.
- In postfix form, the previous value is obtained for use in the expression, and then the operand is modified.

```
x = 42;
y = ++x;
```

In this case both x and y get 43 as x is incremented first and then stored in y.

```
x = 42;
y = x++;
```

In this case x gets 43 and y gets 42 as x is stored in y first and then incremented.

17

18

Increment and Decrement Operator

```
class IncDec {
public static void main(String args[]) {
int a = 1;
int b = 2;
int c;
int d;
c = ++b;
d = a++;
c++;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
}
}
```

18

19

The Bitwise Operators

- Java defines several bitwise operators that can be applied to the integer types, long, int, short, char, and byte. These operators act upon the individual bits of their operands.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

19

20

The Bitwise Logical Operators

- The bitwise logical operators are &, |, ^, and ~. The following table shows the outcome of each operation.

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

- The bitwise complement, the unary NOT operator, ~, inverts all of the bits of its operand.
- For example, the number 42, which has the following bit pattern: 00101010 becomes 11010101 after the NOT operator is applied.

20

21

Binary numbers and 2's complement

- Since the bitwise operators manipulate the bits within an integer, it is important to understand what effects such manipulations may have on a value.
- Specifically, it is useful to know how Java stores integer values and how it represents negative numbers.
- byte x = 42 (is same as 00101010 in binary)
- Java uses an encoding known as two's complement, which means that negative numbers are represented by inverting (changing 1's to 0's and vice versa) all of the bits in a value, then adding 1 to the result.
- For example, –42 is represented by inverting all of the bits in 42, or 00101010, which yields 11010101, then adding 1, which results in 11010110, or –42.
- To decode a negative number, first invert all of the bits, then add 1. For example, –42, or 11010110 inverted, yields 00101001, or 41, so when you add 1 you get 42.

21

22

An example using unary NOT

```
class Example {
public static void main(String args[]) {
byte x = 0b00101010;
byte y = (byte) ~x;
System.out.println("x = " + x);
System.out.println("y = " + y);
}
}
```

22

23

The Bitwise AND

- The AND operator, &, produces a 1 bit if both operands are also 1. A zero is produced in all other cases.

```
        00101010    42
&       00001111    15
        00001010    10
```

```
class BitwiseAND {
public static void main (String[] args) {
int number1 = 42, number2 = 15, result;
result = number1 & number2;
System.out.println(result);}
}
```

23

24

The Bitwise OR

- The OR operator, |, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown here:

```
        00101010    42
|       00001111    15
        00101111    47
```

```
class BitwiseOR {
public static void main(String[] args) {
int number1 = 42, number2 = 15, result;
result = number1 | number2;
System.out.println(result);}
}
```

25

The Bitwise XOR

- The XOR operator, ^, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise, the result is zero.
- The following example shows the effect of the ^.
- The example also demonstrates a useful attribute of the XOR operation. Notice how the bit pattern of 42 is inverted wherever the second operand has a 1 bit. Wherever the second operand has a 0 bit, the first operand is unchanged.

```
      00101010    42
^     00001111    15
      00100101    37
```

Write a program which uses inverting bit property of bitwise xor to take one's complement.

25

26

Taking Two's complement using XOR

```
class Test {
public static void main(String args[]) {
byte x = 0b00101010;
byte y = (byte) (x^0b11111111);
byte z = (byte)  (y + 1);
System.out.println("x = " + x);
System.out.println("y = " + y);
System.out.println("z = " + z);
}
}
```

26

27

The Left Shift

- The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times. It has this general form:

value << num

- Here, num specifies the number of positions to left-shift the value in value
- For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right.
- This means that when a left shift is applied to an int operand, bits are lost once they are shifted past bit position 31. If the operand is a long, then bits are lost after bit position 63.
- Java's automatic type promotions produce unexpected results when you are shifting byte and short values. In order to use shift on byte, you have to do casting to byte again after operation.

27

28

The Left Shift on byte

```
// Left shifting a byte value.
class ByteShift {
public static void main(String args[]) {
byte a = 64, b;
int i;
i = a << 2;
b = (byte) (a << 2);
System.out.println("Original value of a: " + a);
System.out.println("i and b: " + i + " " + b);
}
}
```

28

29

The Left Shift to multiply by 2

```
// Left shifting as a quick way to multiply by 2.

class MultByTwo {
public static void main(String args[]) {
byte num = 32;
num =  (byte) (num << 1);
System.out.println(num);
num =  (byte) (num << 1);
System.out.println(num);}
} } }
```

29

30

The Right Shift

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.

```
int a = 32;
a = a >> 2; // a now contains 8
```

- When a value has bits that are "shifted off," those bits are lost. For example, the next code fragment shifts the value 35 to the right two positions, which causes the two low-order bits to be lost

```
int a = 35;
a = a >> 2; // a still contains 8
```

- When you are shifting right, the top (leftmost) bits exposed by the right shift are filled in with the previous contents of the top bit. This is called sign extension and serves to preserve the sign of negative numbers when you shift them right. For example, –8 >> 1 is –4,

30

31

The Unsigned Right Shift

- The >> operator automatically fills the high-order bit with its previous contents each time a shift occurs. This preserves the sign of the value.
- Sometimes this is undesirable. This situation is common when you are working with pixel-based values and graphics.
- Java's unsigned, shift-right operator, >>>, always shifts zeros into the high-order bit.

int a = -1;

a = a >>> 24;

Here is the same operation in binary form to further illustrate what is happening:

11111111 11111111 11111111 11111111 –1 in binary as an int

>>>24

00000000 00000000 00000000 11111111 255 in binary as an int

31

32

Bitwise Operator Compound Assignments

- All of the binary bitwise operators have a compound form similar to that of the algebraic operators, which combines the assignment with the bitwise operation.
- For example, the following two statements, which shift the value in a right by four bits, are equivalent:

a = a >> 4;

a >>= 4;

- Likewise, the following two statements, which result in a being assigned the bitwise expression a OR b, are equivalent:

a = a | b;

a |= b;

32

33

1

2

Relational Operators

- How do you compare two values, such as whether a value is greater than 0, equal to 0, or less than 0?
- Java provides six relational operators (also known as comparison operators), which can be used to compare two values

Relational Operators

Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
<	<	less than	`radius < 0`	`false`
<=	≤	less than or equal to	`radius <= 0`	`false`
>	>	greater than	`radius > 0`	`true`
>=	≥	greater than or equal to	`radius >= 0`	`true`
==	=	equal to	`radius == 0`	`false`
!=	≠	not equal to	`radius != 0`	`true`

2

3

Examples of Relational Operators

- The result produced by a relational operator is a boolean value. For example, the following code fragment is perfectly valid:

```
int a = 4;
int b = 1;
boolean c = (a < b);
```

- In this case, the result of a<b (which is false) is stored in c.

```
boolean d = (a==b);
```

- This statement checks if a is equal to b? Since it is not true so false is stored in d.

3

4

If Statement

- The if statement is Java's conditional branch statement. It can be used to route program execution through two different paths. Here is the general form of the if statement:

```
if (condition)
        statement1;
else
        statement2;
```

- Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block). The condition is any expression that returns a boolean value. The else clause is optional.
- The if works like this: If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed. In no case will both statements be executed.

4

5

If Statement Example

```
int a =5, b=3;
if(a < b)
        a = 0;
else
        b = 0;
```

Here, if checks whether a is less than b, if it was then a would be set to zero. Otherwise, b would be set to zero. In no case both are set to zero. Since a is not less than b so b is set as zero

5

6

Common Mistakes

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

6

7

If for Multiple Statements

Only one statement can appear directly after the if or the else. If you want to include more statements, you'll need to create a block, as in this fragment:

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
ProcessData();
bytesAvailable -= n;
} else
waitForMoreData();
bytesAvailable = n;
```

7

8

A Simple Program

Write a program which calculates area of a circle.

1- The program must ask user to enter radius of circle.

2- If the radius entered by user is greater than or equal to zero, the program must calculate its area and print it on screen.

3- If radius is less than zero then it should print that "Radius must be greater than zero "and exit the program.

8

9

Code

```
Scanner input = new Scanner (System.in);
System.out.print("Please enter radius of circle: ");
int radius = input.nextInt();
double PI = 3.1416;
if (radius >= 0) {
double area = radius * radius * PI;
System.out.println("The area for the circle of radius " +
radius + " is " + area);
}
else
System.out.println("The radius must be greater than
zero");
```

9

10

Another Simple Program

Write a program which checks if a number is even or odd.

1- The program must ask user to enter an integer.

2- If number is even, program must print, it is an even number.

3- If number is odd, program must print, it is an odd number.

Hint: Use remainder (%) operator to find out if a number is even or odd.
Remainder of an even number is always zero.

10

11

Code

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

System.out.print("Please enter a number (integer): ");
int number = input.nextInt();
if (number % 2 == 0)
System.out.println("This is an even number");
else
System.out.println("This is an odd number");
```

11

12

Nested If Statements

- A nested if is an if statement that is the target of another if or else.
- Example of nested if is given below

int i=10, j=8, k=5;

```
if (i > k) {
    if (j > k)
      System.out.println("i and j are greater than
          k");
}
else
System.out.println("i is less than or equal to k");
```

12

13

Else in Nested If Statements

- When you nest ifs, the main thing to remember is that an else statement always refers to the nearest if statement that is within the same block as the else and that is not already associated with an else. Here is an example:

```
int i=10, j=15, a=20, b=25, c=30, d=35, k=120;

if(i == 10) {
     if(k > 100) a = b; // this if is
     else a = c; // associated with this else
}
else a = d; // this else refers to if(i == 10)
```

13

14

Writing Nested if else Statements

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

14

15

Practice Program

Write a program which tells user which category he/she belongs to

1- The program must ask user to enter age as integer.
2- If age is less than 20 it must print “You are Teenager”.
3- If age is from 20 to 35 it must print “You are Young”.
4- If age is from 36 to 50 it must print “ You are middle aged”
5- If age is above 50 it must print “You are old”

Hint: Use nested if else statements.

15

16

Code

```
Scanner input = new Scanner (System.in);
System.out.print("Please enter age: ");
int age = input.nextInt();

if (age < 20)
System.out.print("You are a Teenager");
else if (age <36)
System.out.print("You are young");
else if (age < 51)
System.out.print("You are middle aged");
else
System.out.print("You are old");
```

16

17

Common Errors using if

```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

(a) Wrong

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b) Correct

Logic error

```
if (radius >= 0);
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

Equivalent

Empty block

```
if (radius >= 0) {};
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

17

18

Common Errors using if

```
if (even == true)
  System.out.println(
    "It is even.");
```

Equivalent

This is better

```
if (even)
  System.out.println(
    "It is even.");
```

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

Equivalent

This is better with correct indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

18

19

Logical Operators

- Logical operators are used to combine several logical conditions to form a compound Boolean expression. Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.

Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

19

20

Practice Program

Write a program using logical operators that checks whether a number entered by the user is between certain values and gives remarks accordingly.

- 80< number && number <=100 — You have got distinction
- 60<= number && number<= 80 — You have passed
- 0<= number && number < 60 — You have failed.
- Otherwise — Invalid Number

(Hint: Use if else and && for checking if both conditions are true).

20

21

Practice Program

- Write a program using logical operators that checks whether a number is divisible by 2 and 3, by 2 or 3, and by 2 or 3 but not both.

(Hint: Use && for checking if both conditions are true, use || for checking if one of the condition is true and use ^ for checking if only one condition is true.

21

22

Code (Part I)

```
import java.util.Scanner;

public class TestBooleanOperators {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int number = input.nextInt();
if (number % 2 == 0 && number % 3 == 0)
System.out.println(number + " is divisible by 2 and
3.");
```

22

23

Code (Part II)

```
if (number % 2 == 0 || number % 3 == 0)
System.out.println(number + " is divisible by 2 or
3");

if (number % 2 == 0 ^ number % 3 == 0)
System.out.println(number + " is divisible by 2 or 3,
but not both.");
}
}
```

23

24

Switch Statement

- The switch statement is Java's multiway branch statement.
- It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.
- It often provides a better alternative than a large series of if-else-if statements.
- The general form of a switch statement is shown on next slide:

24

25

Switch Statement General Form

```
switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
...
case valueN:
// statement sequence
break;
default:
// default statement sequence;
}
```

25

26

Switch Statement Example

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a number: ");
int number = input.nextInt();
switch (number) {
case 1:
System.out.println("Please wait for operator");
break;
case 2:
System.out.println("You are in main menu");
break;
default:
System.out.println("Thank you for using service.");
break;}
```

26

27

Switch Statement Working

- The expression must be of type byte, short, int, or char, String
- Each of the values specified in the case statements must be of a type compatible with the expression and duplicate case values are not allowed.
- The value of the expression is compared with each of the literal values in the case statements. If a match is found, the code sequence following that case statement is executed.
- If none of the constants matches the value of the expression, then the default statement is executed. However, the default statement is optional. If no case matches and no default is present, then no further action is taken.
- The break statement is used inside the switch to terminate a statement sequence.
- When a break statement is encountered, execution branches to the first line of code that follows the entire switch statement. This has the effect of "jumping out" of the switch.

27

28

Switch Statement Example

```
import java.util.Scanner;
public class TestSwitch {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter day number: ");
int day = input.nextInt();
switch (day) {
case 1:
case 2:
case 3:
case 4:
case 5:
System.out.println("Weekday");
break;
case 0:
case 6: System.out.println("Weekend");
}}}
```

28

Example using String

29

```
Scanner input = new Scanner(System.in);
System.out.print("Please enter a character from A-Z: ");
String str= input.nextLine();
switch (str) {
   case "A":
      System.out.println("CaseA");
         break;
   case "B":
      System.out.println("CaseB");
         break;
   case "C":
      System.out.println("CaseC");
      break;
   default:
      System.out.println("Default");
}
```

29

Practice Problem

30

Write a switch statement that displays Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, if day is 0, 1, 2, 3, 4, 5, 6, accordingly. Get Day number from user as input in the form of an integer.

30

Calculator Using Switch (I)

31

```
char operator;
double number1, number2, result;

Scanner input = new Scanner(System.in);

System.out.print("Enter operator (either +, -, * or /): ");

operator = input.next().charAt(0);

System.out.print("Enter number1 and number2 respectively: ");
number1 = input.nextDouble();
number2 = input.nextDouble();

switch (operator) {

case '+':
      result = number1 + number2;
       System.out.print(number1 + "+" + number2 + " = " + result);
       break;
```

31

Calculator Using Switch (II)

32

```
case '-':
          result = number1 - number2;
          System.out.print(number1 + "-" + number2 + " = " + result);
          break;
case '*':
          result = number1 * number2;
          System.out.print(number1 + "*" + number2 + " = " + result);
          break;
case '/':
          result = number1 / number2;
          System.out.print(number1 + "/" + number2 + " = " + result);
          break;
default:
          System.out.println("Invalid operator!");
          break;
}
```

32

33

The ? Operator

- Java includes a special ternary (three-way) operator that can replace certain types of if-then-else statements.
- This operator is the ?. It can seem somewhat confusing at first, but the ? can be used very effectively once mastered.
- The ? has this general form:

```
expression1 ? expression2 : expression3;
```

- Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.
- The result of the ? operation is that of the expression evaluated. Both expression2 and expression3 are required to return the same type, which can't be void.
- Here is an example of the way that the ? is employed:

```
ratio = (denom == 0 ? 0 : num / denom);
```

33

34

Example of the ? Operator

```
if (x > 0)
y = 1;
else
y = -1;
```

- Alternatively, as in the following example, you can use a conditional expression to achieve the same result.

```
y = (x > 0) ? 1 : -1;
```

- Another Example

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

34

Practice Problem

Please get age from user using Scanner class and get ticket price based on age. If age is greater than or equal to 16, ticket price is 20 otherwise 10. The if condition is given below.

```
if (age >= 16)
ticketPrice = 20;
else
ticketPrice = 10;
```

Rewrite the following if statements using the conditional ? operator.

35

36

IN1044

Introduction to Programming

Instructor
Dr. Muhammad Waqar
muhammad.aslam@purescollege.ca

1

2

Math Class

- Java provides many useful methods in the Math class for performing common mathematical functions.
- A method is a group of statements that performs a specific task.
- Java provides many mathematical functions in Match class e.g. pow(a, b) method to compute a^b, the random() method for generating a random numbers.
- This lecture introduces useful methods in the Math class which can mainly be categorized as trigonometric methods, exponent methods, and service methods.
- Math class is automatically imported into Java and you do not need to explicitly import it.

2

3

Trigonometric Methods

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

- All methods of Math class can be used by appending Math. before method name e.g. Math.sin(radians), Math.toRadians(degree)

3

4

Exponent Methods

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

4

5

Practice Problems

- Write a program which asks user to enter a number calculates square root of this number. The square root should be displayed at output.
- Write a program which asks user to enter a number and also ask user which power of this number he/she wants to calculate. The required power of number should be calculated and displayed at output. (Hint: use Math.pow(a,b)

5

6

The Rounding Methods

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

6

7

Practice Problems

- Write a program which asks user to enter radius of a circle and calculates its area. The area must be rounded to nearest integer.

7

8

Min, Max and Abs Methods

- The min and max methods return the minimum and maximum numbers of two numbers (int, long, float, or double).
- For example, max(4.4, 5.0) returns 5.0, and min(3, 2) returns 2.
- The abs method returns the absolute value of the number (int, long, float, or double).
- For example,

Math.max(2, 3) returns 3

Math.max(2.5, 4) returns 4.0

Math.min(2.5, 4.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

8

9

Practice Problem

- Write a program which asks user to enter two numbers (one by one) and prints which number is bigger out of the two using Math class.

9

10

The Random Method

- This method generates a random double value greater than or equal to 0.0 and less than 1.0
- 0.0 <= Math.random() <1.0).
- You can use it to write a simple expression to generate random numbers in any range.
- (int)(Math.random() * 10) Returns a random integer between 0 and 9.
- 50 + (int)(Math.random() * 50) Returns a random integer between 50 and 99.

10

11

Practice Problem

- Write a program which generates output of a dice randomly (a number between 1 and 6). Asks user to guess that number and if the guess is correct it should print that you have guessed right otherwise it should print that your guess is wrong and should also print both numbers.

11

12

The String Class

- The char type represents only one character. To represent a string of characters, use the data type called String. For example, the following code declares message to be a string with the value "Welcome to Java".
- String message = "Welcome to Java";
- String is a predefined class in the Java library, just like the classes System and Scanner.
- The String type is not a primitive type. It is known as a reference type.
- Any Java class can be used as a reference type for a variable.
- The variable declared by a reference type is known as a reference variable that references an object. Here, message is a reference variable that references a string object with contents Welcome to Java.

12

13

Methods in the String Class

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

13

14

Practice Problem

Write a program which declares the following string

String message = "Welcome to Java";

Use length() method to print the length of string. Then print the character at index 11 using charAt() method. Convert this string to upper and lower case using toUpperCase() and toLowerCase() methods. Declare another string

String message1 = “You are learning Strings”

Now combine message with message 1 using concat() method and also using + sign.

14

15

Reading One Word as String from Console

To read one string from the console, invoke the next() method on a Scanner object. A string finishes whenever a space is pressed. For example, the following code reads three strings from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

15

16

Reading Complete String from Console

- You can use the nextLine() method to read an entire line of text.
- The nextLine() method reads a string that ends with the Enter key pressed. For example, the following statements read a line of text.

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

16

17

Reading Character from Console

- To read a character from the console, use the nextLine() method to read a string and then invoke the charAt(0) method on the string to return a character.
- For example, the following code reads a character from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

17

18

Comparing Strings

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if s1 is a substring in this string.

18

19

Comparing Strings

Declare two strings

String s1 = "Welcome to Java";

String s2 = “welcome to java";

Write a program which compares s1 and s2 and prints results using first two methods given on previous slide.

19

20

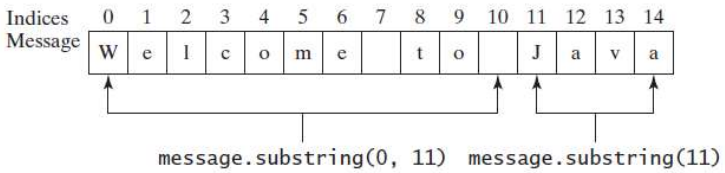
Obtaining Substrings

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex` – 1, as shown in Figure 4.2. Note that the character at `endIndex` is not part of the substring.

String message = "Welcome to Java";

String message = message.substring(0, 11) + "HTML";

The string message now becomes Welcome to HTML.



20

21

Practice Problem

Write a Java program which prints the following statement.

"The quick brown fox jumps over the lazy dog."

Ask the user to specify two integer indexes. Tell the user that the second index cannot be greater than the length of the string and then extract substring for those indexes and print that substring.

21

22

Finding a Substring in a String

Method	*Description*
`index(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

22

23

Example of Finding Substrings

"Welcome to Java".indexOf('W') returns 0.

"Welcome to Java".indexOf('o') returns 4.

"Welcome to Java".indexOf('o', 5) returns 9.

"Welcome to Java".indexOf("come") returns 3.

"Welcome to Java".indexOf("Java", 5) returns 11.

"Welcome to Java".indexOf("java", 5) returns -1.

23

24

Practice Problem

Write a program which asks user to enter First and Surname and stores it in one string. Use substring methods to divide the First and Surname.

Hint: Find the index of space (“ ”) and select substrings before and after that index.

24

Conversion between Strings and Numbers

You can convert a numeric string into a number. To convert a string into an int value, use the Integer.parseInt method, as follows:

```
int intValue = Integer.parseInt(intString);
```

where intString is a numeric string such as "123".

To convert a string into a double value, use the Double.parseDouble method, as follows:

```
double doubleValue = Double.parseDouble(doubleString);
```

where doubleString is a numeric string such as "123.45".

If the string is not a numeric string, the conversion would cause a runtime error.

You can convert a number into a string, simply use the string concatenating operator as follows:

```
String s = number + "";
```

25

26

IN1044

Introduction to Programming

Instructor
Dr. Muhammad Waqar
muhammad.aslam@purescollege.ca

1

Iteration Statements (Loops)

- A loop repeatedly executes the same set of instructions until a termination condition is met.
- Suppose that you need to display a string (e.g., Welcome to Java!) a hundred times. It would be tedious to have to write the following statement a hundred times:

- System.out.println("Welcome to Java!");
- System.out.println("Welcome to Java!");
- ...
- System.out.println("Welcome to Java!");

2

3

The While Loop

- A while loop executes statements repeatedly while the condition is true.
- The syntax for the while loop is:

```
while (loop-continuation-condition) {
// Loop body
}
```

- The part of the loop that contains the statements to be repeated is called the loop body.
- A one-time execution of a loop body is referred to as an iteration (or repetition) of the loop.

3

4

The While Loop

- Each loop contains a loop-continuation-condition, a **boolean** expression that controls the execution of the body.
- The condition can be any **boolean** expression.
- This condition is evaluated each time to determine if the loop body is to be executed.
- If its evaluation is true, the loop body is executed; if its evaluation is false, the entire loop terminates and the program control turns to the statement that follows the while loop.
- The curly braces are unnecessary if only a single statement is being repeated.

4

5

The while Loop Example

```
int count = 0;                                  // variable initialization
while (count < 100) {                           // checking condition
System.out.println("Welcome to Java!");         // loop body
count++;                                        // incrementing count
}
```

This Loops will print “Welcome to Java!” 100 times.

5

6

Another While Loop Example

```
int sum = 0, i = 1;
while (i <= 10) {
sum = sum + i;
i++;
}
System.out.println("sum is " + sum);
```

This while loop sums ups numbers from 1 to 10 and prints result.

6

7

A common mistake

- A common mistake using loops is to use a boolean condition which is always true and the loops runs forever. For example the following code runs forever

```
int sum = 0, i = 1;
while (i <= 10) {
sum = sum + i;
}
```

- Make sure that the loop-continuation-condition eventually becomes false so that the loop will terminate. If your program takes an unusually long time to run and does not stop, it may have an infinite loop. If you are running the program from the command window, press CTRL+C to stop it. If you are using NetBeans use a stop sign on left side of output window.

7

8

Loop Design Strategy

Writing a correct loop is not an easy task for novice programmers. Consider three steps when writing a loop.

Step 1: Identify the statements that need to be repeated.

Step 2: Wrap these statements in a loop like this:

```
while (true) {
Statements;
}
```

Step 3: Code the loop-continuation-condition and add appropriate statements for controlling the loop.

```
while (loop-continuation-condition) {
Statements;
Additional statements for controlling the loop;
}
```

8

9

Simple Problem

Write a while loop which keeps on printing “Hello” as long as user enters ‘y’ or ‘Y’ and exits program printing “bye” if user enters anything else

9

10

Code

```
public static void main(String args[]) {

Scanner input = new Scanner(System.in);
String s1 = "y";
while (s1.equals("y")){
   System.out.println("Hello");
   System.out.println("Do you want to print hello again?.");
   s1 = input.nextLine();
   s1 =s1.toLowerCase();
}
}
```

10

11

Practice Problem

- Please write a program which prints number from 1 to 5 using while loop. The output should be

- Count = 1
- Count = 2
- Count = 3
- Count = 4
- Count = 5

11

12

Practice problem

- Please write a program using while which prints even numbers from 0 to 20

- 0
- 2
- 4
- 6
- 8
- .
- .
- .
- 20

12

13

Practice Problem

Write a program which calculates sum of the numbers entered by user.

Specifications:

1- Please ask user to enter a number to get sum and enter 0 to end the program.

2- Keep asking user to enter more numbers and keep calculating sum until user enters 0.

3- When user enters 0, print the sum and exit the program.

13

14

Code

```
public static void main(String args[]) {
System.out.println("This program sum the numbers entered by user.");
Scanner input = new Scanner(System.in);
int count = 0;
int check = 0;
int sum =0;
while (check == 0){
   System.out.println("Please enter a number to sum or enter 0 to exit.");
   int number = input.nextInt();
   if (number!=0){
      sum += number;
      count++;}
   else
      check =1;}
   System.out.println("The sum of " + count + " numbers is " + sum);
  }
```

14

15

do while Loop

- A do-while loop is the same as a while loop except that it executes the loop body first and then checks the loop continuation condition.
- The do-while loop is a variation of the while loop. Its syntax is:

```
do {
// Loop body;
Statement(s);
} while (loop-continuation-condition);
```

15

16

do while Loop

- If the conditional expression controlling a while loop is initially false, then the body of the loop will not be executed at all.
- Sometimes it is desirable to execute the body of a loop at least once, even if the conditional expression is false to begin with.
- In other words, there are times when you would like to test the termination expression at the end of the loop rather than at the beginning. A loop that does just that is the do-while.
- The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.
- You can write a loop using either the while loop or the do-while loop. Sometimes one is a more convenient choice than the other.

16

17

Example of do while

This programs calculates sum of numbers entered by user and exits if 0 is entered by user.

```
Scanner input = new Scanner(System.in);
int data;
int sum = 0;
do {
System.out.print("Enter an integer (the input ends if it is 0): ");
data = input.nextInt();
sum += data;
} while (data != 0);
System.out.println("The sum is " + sum);
```

17

18

Guess a number problem

- Write a program which generates a random number between 1 and 10. It asks the user to guess this number. If the answer is right, It should display that you have guessed in these many attempts.
- If the answer is wrong, it should tell the user that your answer is wrong. Please try again.
- If the user does not answer right in 5 attempts then it should exit the program and should tell the user that you have lost and should display the random number.
- Use while loop to iteratively run the program.

18

19

Code

```
Scanner input = new Scanner(System.in);
System.out.println("This is guess the number problem. \n Computer will generate a number between 1 and 10 and you will have to guess ");
int number = 1 + (int)(Math.random() * 10);
int count = 1;
while (count < 6) {
   System.out.print("Please guess a number between 1 and 10: ");
   int guess = input.nextInt();
   if (guess == number ){
      System.out.println("You have guessed right in "+ count + " attempts.");
      break;}
   else{
      System.out.println("You have guessed wrong. You have "+ (5-count) + " attempts left.");
      count++;}
}
System.out.println("The random number was: " + number);
```

19

20

The for loop

- The for loop statement starts with the keyword for, followed by a pair of parentheses enclosing the control structure of the loop.
- This structure consists of initial-action, loop-continuation-condition, and action-after-each-iteration.
- The control structure is followed by the loop body enclosed inside braces.
- The initial-action, loop continuation- condition, and action-after-each-iteration are separated by semicolons.

```
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
// Loop body;
}
```

20

21

The for loop example

- The following loop prints “Welcome to Java!” 100 times.

```
for (int i = 0; i < 100; i++) {
System.out.println("Welcome to Java!");
}
```

- initial-action: int i=0;
- loop-continuation-condition: i<100
- action-after-each-iteration: i++

21

22

More on for loop

- The initial-action in a for loop can be a list of zero or more comma-separated variable declaration statements or assignment expressions. For example:

```
for (int i = 0, j = 0; (i + j) < 10; i++, j++) {
// Do something
}
```

- The conditional expression does not always need to involve declared variable

```
boolean done = false;
for(int i=1; !done; i++) {
// ...
if(interrupted()) done = true;
}
```

22

23

Another Example

The following for loop sums up numbers from 0 to 1000.

```
long sum = 0;
for (int i = 0; i <= 1000; i++)
        sum = sum + i;
System.out.println("Sum is: " + sum);
```

If there are no {} after for loop then only one statement after for is executed.

23

24

Which loop to use?

- You can use a for loop, a while loop, or a do-while loop, whichever is convenient.
- The while loop and for loop are called pretest loops because the continuation condition is checked before the loop body is executed.
- The do-while loop is called a posttest loop because the condition is checked after the loop body is executed.
- The three forms of loop statements—while, do-while, and for—are expressively equivalent and you can write a loop in any of these three forms.
- Use the loop statement that is most intuitive and comfortable for you.
- In general, a for loop may be used if the number of repetitions is known in advance, as, for example, when you need to display a message a hundred times.
- A while loop may be used if the number of repetitions is not fixed, as in the case of reading the numbers until the input is 0.

24

25

Practice Problem

- Write a program which prints number and square of those number from 1 to 10 using for and while loop. A sample output is shown below

1	1
2	4
3	9
10	100

25

26

Practice Problem

Write a program using for loop to print the numbers between 1 and 10, along with an indication of whether the number is even or odd, like this:

1 is odd

2 is even

3 is odd

(Hint: use an *if statement* to decide if the remainder that results by dividing the number by 2 is 0. There is a *modulus operator* %, which can be used to achieve this, where x%y is equal to the remainder of the first operand (x) divided by the second (y)).

26

27

Nested loops

- Nested loops consist of an outer loop and one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered, and started anew.

```
class Nested {
public static void main(String args[]) {
int i, j;
for(i=0; i<10; i++) {
     for(j=0; j<10; j++)
          System.out.print("*");
     System.out.println();
} } }
```

27

28

Practice problem

- Write a program which gives the following output. Use nested for loop for your program

```
*
**
***
****
*****
```

28

29

Practice problem code

```
class Nested {
public static void main(String args[]) {
int i, j;
for(i=1; i<=5; i++) {
     for(j=1; j<=i; j++)
          System.out.print("*");
     System.out.println();
} } }
```

29

30

Practice problem

- Write a program which gives the following output. Use nested for loop for your program

```
*****
****
***
**
*
```

30

31

Practice problem

- Write a program which gives the following output. Use nested for loop for your program

```
    *
   **
  ***
 ****
*****
```

32

Home Work problem

- Write a program which gives the following output

```
    *
  ***
*****
  ***
    *
```

33

Practice Problem

Write a program which prints even numbers from 0 to 20 (using for loop) and then asks user do you want to print again? (using while loop) if user enter 'y' or 'Y' then the program must print even numbers from 0 to 20 again and should ask the user his/her choice again.

33

34

Jump Statements (break and continue)

- Java supports three jump statements: break, continue, and return. These statements transfer control to another part of your program.
- break is used to terminate a statement sequence in a switch statement. It can also be used to exit a loop.
- By using break, you can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
- When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

34

35

Example of break statement

```
// Using break to exit a loop.
class BreakLoop {
public static void main(String args[]) {
for(int i=0; i<100; i++) {
        if(i == 10)
                break; // terminate loop if i is 10
        System.out.println("i: " + i);
}
System.out.println("Loop complete.");
}
}
```

35

36

break inside nested loops

- When used inside a set of nested loops, the break statement will only break out of the innermost loop

```
class BreakLoop3 {
public static void main(String args[]) {
for(int i=0; i<3; i++) {
        System.out.print("Pass " + i + ": ");
        for(int j=0; j<100; j++) {
                if(j == 10) break; // terminate loop if j is 10
                System.out.print(j + " ");
        }
System.out.println();
}
System.out.println("Loops complete.");
}}
```

36

37

continue statement

- Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. The continue statement performs such an action.
- In while and do-while loops, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.
- In a for loop, control goes first to the iteration portion of the for statement and then to the conditional expression. For all three loops, any intermediate code is bypassed.

37

38

Example of continue statement

```
class Continue {
public static void main(String args[]) {
for(int i=0; i<10; i++) {
        System.out.print(i + " ");
        if (i%2 == 0)
                continue;
        System.out.println("");
}
}
}
```

38

39

continue to a specific label

Continue may specify a label to describe which enclosing loop to continue. Here is an example program that uses continue to print a triangular multiplication table for 0 through 9.

```
class ContinueLabel {
public static void main(String args[]) {
outer: for (int i=0; i<10; i++) {
        for(int j=0; j<10; j++) {
                if(j > i) {
                        System.out.println();
                        continue outer; }
                System.out.print(" " + (i * j));
        }
}
}}
```

39

40

Practice problem: Checking Palindrome

- A string is a palindrome if it reads the same forward and backward.
- The words “mom,” “dad,” and “noon,” “madam”, “deleveled’ for instance, are all palindromes.
- The problem is to write a program that prompts the user to enter a string and reports whether the string is a palindrome.

40

41

Practice problem code (Part I)

```
import java.util.Scanner;
public class Palindrome {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter a string: ");
String s = input.nextLine();
int low = 0; // The index of the first character in the string
int high = s.length() - 1; // The index of the last character in
                           // the string
boolean isPalindrome = true;
```

41

42

Practice problem code (Part II)

```
while (low < high) {
      if (s.charAt(low) != s.charAt(high)) {
            isPalindrome = false;
            break;}
      low++;
      high--;
}
if (isPalindrome)
      System.out.println(s + " is a palindrome");
else
      System.out.println(s + " is not a palindrome");
}}
```

42

43

Practice Problem

- An integer greater than 1 is prime if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.
- The problem is to ask the user to enter a number and print out at output whether this number is a prime number.

43

44

Home Work Problems

1) Ask the user to enter a number and count the number of positive numbers, negative numbers and zeros entered by the user. The program should ask user after every number do you want to enter another number? And should continue if user enters 'y' or 'Y'. (while and for loop)
2) Print multiplication tables from 1 to 10 using nested for loops. (for and for loop)
3) Convert a table of pound to Kgs from 1 to user defined value and ask user do you want to print another table? (while and for loop)
4) Ask student name and students marks to user defined number of students and display the student name with highest marks. (You can use arrays) (for loop)
5) Ask user a number and calculate its factorial and ask user do you want to calculate factorial of another number? (while and for loop)
6) Find factors of a user defined number, display its factors and ask user do you want to calculate factors of another number? (while and for loop)

44

45

1

2

Methods

- Methods can be used to define reusable code and organize and simplify coding.
- A method is a collection of statements grouped together to perform an operation.
- Previously you have used predefined methods such as System.out.println, Math.pow, and Math.random. These methods are defined in the Java library.

2

3

Defining Methods

- A method definition consists of its method name, parameters, return value type, and body.
- The syntax for defining a method is as follows:

```
modifier returnValueType methodName(list of parameters) {
// Method body;
}
```

3

4

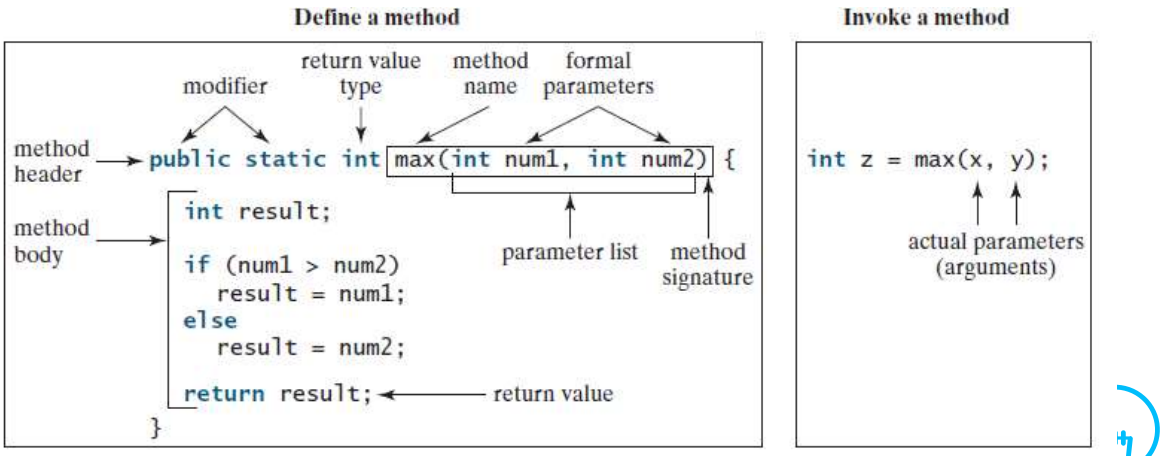
Different parts of a method

- The method header specifies the modifiers, return value type, method name, and parameters of the method. The static modifier is used for all the methods in this
- A method may return a value. The returnValueType is the data type of the value the method returns.
- Some methods perform desired operations without returning a value. In this case, the returnValueType is the keyword void. For example, the returnValueType is void in the main method.
- The variables defined in the method header are known as formal parameters or simply parameters.
- A parameter is like a placeholder: when a method is invoked, you pass a value to the parameter. This value is referred to as an actual parameter or argument. Parameters are optional; that is, a method may contain no parameters.

4

5

Example of a method



5

6

Calling a method

- Calling a method executes the code in the method.
- In a method definition, you define what the method is to do.
- To execute the method, you have to call or invoke it.
- There are two ways to call a method, depending on whether the method
- returns a value or not.
- If a method returns a value, a call to the method is usually treated as a value. For example,
- int larger = max(3, 4);

  calls max(3, 4) and assigns the result of the method to the variable larger
- If a method returns void, a call to the method must be a statement. For example, the method println returns void.

6

7

Calling a method

- When a program calls a method, program control is transferred to the called method.
- A called method returns control to the caller when its return statement is executed or when its method ending closing brace is reached.
- Methods enable code sharing and reuse. The max method can be invoked from any class.
- If you call a method in the same class where it has been defined, it can be used directly by name e.g. max(i,j)
- If you create a new class, you can invoke the max method using ClassName.methodName (i.e., TestMax.max(i,j)).

7

8

Example

```
public class TestMax {
public static void main(String[] args) {
int i = 5;
int j = 2;
int k = max(i, j);
System.out.println("The maximum of " + i +
" and " + j + " is " + k);
}

public static int max(int num1, int num2) {
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;}}
```

8

An example of void method

- A void method does not have a return type as it does not return a value

```
public class TestVoidMethod {

public static void main(String[] args) {

System.out.print("The grade is ");
printGrade(78.5);
System.out.print("The grade is ");
printGrade(59.5);
}
}
```

9

An example of void method


```
public static void printGrade(double score) {
if (score >= 90.0) {
       System.out.println('A');}
else if (score >= 80.0) {
       System.out.println('B');}
else if (score >= 70.0) {
       System.out.println('C');}
else if (score >= 60.0) {
       System.out.println('D');}
else {
       System.out.println('F');
}
}
}
```

10

11

return statement in void method

- A return statement is not needed for a void method, but it can be used for terminating the method and returning to the method's caller.
- The syntax is simply

  return;
- This is not often done, but sometimes it is useful for circumventing the normal flow of control in a void method. An example is given on next slide.

11

12

Example of return statement in void method

```
public static void printGrade(double score) {
if (score < 0 || score > 100) {
System.out.println("Invalid score");
return;}
if (score >= 90.0) {
       System.out.println('A');}
else if (score >= 80.0) {
       System.out.println('B');}
else if (score >= 70.0) {
       System.out.println('C');}
else if (score >= 60.0) {
       System.out.println('D');}
else {
       System.out.println('F');
}}
```

12

13

Argument passing by values

- When calling a method, you need to provide arguments, which must be given in the same order as their respective parameters in the method signature. This is known as parameter order association. For example, the following method prints a message n times:

```
public static void nPrintln(String message, int n) {
for (int i = 0; i < n; i++)
System.out.println(message);}
```

- You can use nPrintln("Hello", 3) to print Hello three times.
- The nPrintln("Hello", 3) statement passes the actual string parameter Hello to the parameter message, passes 3 to n, and prints Hello three times.
- However, the statement nPrintln(3, "Hello") would be wrong. The data type of 3 does not match the data type for the first parameter, message, nor does the second argument, Hello, match the second parameter, n.

13

14

Argument passing by values

- The arguments must match the parameters in order, number, and compatible type, as defined in the method signature.
- Compatible type means that you can pass an argument to a parameter without explicit casting, such as passing an int value argument to a double value parameter.
- When you invoke a method with an argument, the value of the argument is passed to the parameter. This is referred to as pass-by-value.
- If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter.
- The variable is not affected, regardless of the changes made to the parameter inside the method.

14

15

Argument passing by values example

```
public class Increment {
public static void main(String[] args) {
 int x = 1;
 System.out.println("Before the call, x is " + x);
 increment(x);
 System.out.println("After the call, x is " + x);
 }
 public static void increment(int n) {
 n++;
 System.out.println("n inside the method is " + n);
 }
 }
```

15

16

Practice Problem

- Write a program which defines two methods, max and min.
- Both these methods should accept two integers as input and should return, maximum and minimum value in a return statement.
- Call these methods in main() method and print out outputs returned by these methods.

16

17

Method Overloading

- Overloading methods enables you to define the methods with the same name as long as their signatures are different.
- The max method that was used earlier works only with the int data type. But what if you need to determine which of two floating-point numbers has the maximum value?
- The solution is to create another method with the same name but different parameters

17

18

Method Overloading

```
public static double max(double num1, double num2) {
if (num1 > num2)
        return num1;
else
        return num2;}
```

- If you call max with int parameters, the max method that expects int parameters will be invoked;
- if you call max with double parameters, the max method that expects double parameters will be invoked.
- This is referred to as method overloading; that is, two methods have the same name but different parameter lists within one class.
- The Java compiler determines which method to use based on the method signature.

18

19

Method Overloading Example (Part I)

```
public class TestMethodOverloading {
public static void main(String[] args) {
System.out.println("The maximum of 3 and 4 is " + max(3, 4));
System.out.println("The maximum of 3.0 and 5.4 is "+ max(3.0,
5.4));
System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
+ max(3.0, 5.4, 10.14));}

public static int max(int num1, int num2) {
if (num1 > num2)
      return num1;
else
      return num2;
}
```

19

20

Method Overloading Example (Part I)

```
public static double max(double num1, double num2) {
if (num1 > num2)
      return num1;
else
      return num2;
 }

public static double max(double num1, double num2, double num3) {
return max(max(num1, num2), num3);
}
}
```

20

21

Practice Problem I

Write a program which defines a method to calculate multiplication of two numbers e.g. multiplyInteger.

Call this method in the main method and calculate multiplication of two numbers.

21

22

Practice Problem II

Write a program which defines three overloaded methods to calculate multiplication of numbers.

First method should accept two values and should return multiplication of two numbers.

Second and third method should have three and four numbers and should give the multiplication result at output respectively.

22

Practice Problem III

Write a Java method to count all vowels in a string.

Define a method which counts vowels in any string.

Ask the user to enter a string in main() method.

Call countvowels method in main() method and return value to main()

Print out the number of vowels in string.

23

24