

CSE2101: Object Oriented Programming-II (Java)

Lecture 2

Java – Data Types, Variables, Expression

Typical Java Development Environment

3

- Java programs normally go through five phases.
 - Edit – create/edit the source code
 - Compile – compile the source code
 - Load – load the compiled code
 - Verify – check against security restrictions
 - Execute – execute the compiled code

Phase 1: Creating a Program

4

- Any text editor or Java IDE (Integrated Development Environment) can be used to develop Java programs.
- Java source-code file names must end with the **.java** extension.
- Some popular Java IDEs are NetBeans, jEdit, Eclipse, JBuilder, JCreator etc.

Phase 2: Compiling a Java Program into Bytecodes

5

- javac Welcome.java
 - Compiles the source file “Welcome.java” (and other files if necessary), transforms the Java source code into bytecodes and places the **bytecodes** in a file named “Welcome.class”.
 - It searches the file “Welcome.java” in the current directory and in the directories listed in the **PATH** environment variable.
- Bytecodes
 - They are not machine language binary code.
 - They are independent of any particular microprocessor or hardware platform.
 - They are platform-independent instructions.
 - Another entity or **interpreter** is required to convert the bytecodes into machine codes that the underlying microprocessor understands.
 - This is the job of the **JVM (Java Virtual Machine)**

JVM (Java Virtual Machine)

6

- It is a part of the JDK and the foundation of the Java platform.
- It can be installed separately or with JDK.
- A **virtual machine (VM)** is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with the VM.
- One of the main contributors for the **slowness** of Java programs compared to compiled machine language programs (i.e. C, C++, Pascal etc.).

JVM (Java Virtual Machine) (contd.)

7

- It is the JVM that makes Java a portable language.
- The same bytecodes can be executed on any platform containing a compatible JVM.
- JVM is available for Windows, Unix, Linux and Solaris.
- The JVM is invoked by the “**java**” command.
 - java Welcome
 - It searches the class “Welcome” in the current directory and in the directories listed in the **CLASSPATH** environment variable and executes the “main” method of class “Welcome”.
 - It issues an error if it cannot find the class “Welcome” or if class “Welcome” does not contain a method called “main” with proper signature (more on this will be discussed later)

Phase 3: Loading a Program into Memory 8

- ☐ One of the components of the JVM is the **class loader**.
- ☐ The class loader takes the .class files containing the programs bytecodes and transfers them to primary memory (RAM).
- ☐ The class loader also loads any of the .class files provided by Java that our program uses.
- ☐ The .class files can be loaded from a disk on our system or over a network (another PC in our LAN, or the Internet).

Phase 4: Bytecode Verification

9

- ☐ Another component of the JVM is the **bytecode verifier**.
- ☐ Its job is to ensure that bytecodes are valid and do not violate Java's security restrictions.
- ☐ This feature helps to prevent Java programs arriving over the network from damaging our system.
- ☐ Another contributor for making Java programs slow.

Phase 5: Execution

10

- Now the actual execution of the program begins.
- Bytecodes are converted to machine language suitable for the underlying OS and hardware.
- Recent JVM performs **just-in-time (JIT) compilation** to make program execution faster.
- So, we can say that Java programs actually go through two compilation phases –
 - Source code -> bytecodes
 - Bytecodes -> machine language

Identifiers, Keyword, Statements

Identifiers

12

- ❑ Identifiers are names for variables, classes, methods etc.
- ❑ Good ones are compact, but indicate what they stand for
 - radius, width, height, length
- ❑ Java is case sensitive (so as identifier).
- ❑ Rules:
 - May contain upper case, lower case letters, numbers, underscore, dollar sign.
 - Must not begin with a number.

Keywords

13

- Some words are reserved, and can't be used as identifiers

```
// Authors: J. P. Cohoon and J. W. Davidson  
// Purpose: display a quotation in a console window
```

```
public class DisplayForecast {  
  
    // method main(): application entry point  
    public static void main(String[] args) {  
        System.out.print("I think there is a world market for");  
        System.out.println(" maybe five computers.");  
        System.out.println(" Thomas Watson, IBM, 1943.");  
    }  
}
```

Capitalization

14

- Case matters!
- public ≠ Public ≠ PUBLIC
 - This is different that FORTRAN and BASIC
 - This is the same as C/C++

Statements

15

- A statement in Java is (usually) a single line
 - Example: `System.out.println ("Hello world!");`
- All statements must end with a semi-colon (like C)

Data types, Variables

Data Types

17

- Java is a “strong typed language”
 - Each variable has a declared type.

```
float x;    //x is a variable  
x = 13.2;
```

- There are two kinds of data-types in Java:
 - Primitive types.
 - Classes (will be discussed later).

Java Primitive Types

18

- There are 8 primitive types in Java.
- Integer types:

byte	An 8-bit signed integer.
short	A 16-bit signed integer.
int	A 32-bit signed integer.
long	A 64-bit signed integer.

Java Primitive Types (Cont.)

19

□ Floating point types:

Float	A 32-bit IEEE floating point.
-------	-------------------------------

double	A 64-bit IEEE floating point.
--------	-------------------------------

□ Other types:

boolean	Either true or false.
---------	-----------------------

char	A 16-bit Unicode character.
------	-----------------------------

Primitive variable types

20

☐ Java has 8 (or so) primitive types:

- float real numbers
- double real numbers
- boolean two values: true and false
- char a single character
- byte
- short integer numbers
- int
- long

☐ Also the “void” type

Primitive real (floating-point) types

21

- A float takes up 4 bytes of space
 - Has 6 decimal places of accuracy: 3.14159

- A double takes up 8 bytes of space
 - Has 15 decimal places of accuracy:
3.14159265358979

- Always use doubles
 - It will save you quite a headache!

Primitive integer types

22

- Consider a byte:

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

- A Java byte can have values from -128 to 127

- From -2^7 to 2^7-1

- C/C++ has unsigned versions; Java does not

- What would be the result for the following program?

```
byte a=127;  
a+=1;  
System.out.println(a);
```

- The Result will be: -128

Primitive integer types

23

Type	Bytes	Minimum value	Maximum value
byte	1	$-2^7 = -128$	$2^7 - 1 = 127$
short	2	$-2^{15} = -32,768$	$2^{15} - 1 = 32,767$
int	4	$-2^{31} = -2,147,483,648$	$2^{31} - 1 = 2,147,483,647$
long	8	$-2^{63} = -9,223,372,036,854,775,808$	$2^{63} - 1 = 9,223,372,036,854,775,807$

Defining and initializing variables

24

- Variables must be declared before use
- Initialization:
 - `int a = 30; //initialization`
- Assignment:
 - `long b;`
 - `b=-20; //assignment`

Variable initialization

25

- Consider the following code:

```
int x;  
System.out.println(x);
```

- What happens?
- Error message:
 - variable x might not have been initialized
- Java requires you to give x a value before you use it

Printing variables

26

- To print a variable to the screen, put it in a `System.out.println()` statement:
 - `int x = 5;`
 - `System.out.println ("The value of x is " + x);`
- Important points:
 - Strings are enclosed in double quotes
 - If there are multiple parts to be printed, they are separated by a plus sign

Primitive character type

27

- All characters have a integer equivalent
 - '0' = 48
 - '1' = 49
 - 'A' = 65
 - 'a' = 97
- Thus, you can refer to 'B' as 'A'+1
- Example:
 - `char var='a';` or, `char var=97;`
 - `var++;` //now, `var='b'`
- There are no negative char. So the range of char is 0-65536

Primitive boolean type

28

- The boolean type has only two values:
 - true
 - false
- Example:
 - `boolean var=true;`
- There are boolean-specific operators
 - `&&` is and
 - `||` is or
 - `!` is not
 - etc.

Literals

29

- Integer literals:
 - Octal base: 034
 - Hexadecimal base: 0x3A
- Floating point literals:
 - Standard notation: 42.4362
 - Scientific notation: 424362E-4
- Boolean Literals:
 - The values of true and false do not convert into any numerical representation. (so, true \neq 1)
- Character Literals:
 - `\n` – New line, `\t` – tab, `\"` – double quote, `\'` – single quote.
 - Enclosed by a single quote. `'a'`, `'\n'`
- String Literals:
 - Enclosing by a pair of double quotes.
 - `"hello world"`

Constants

30

- Consider the following:

```
final int x = 5;
```

- The value of x can NEVER be changed!
 - The value assigned to it is “final”
- This is how Java defines constants

Take Input and Print output

I/O streams

32

- **System.out**
 - Prints to standard output
 - Equivalent to “cout” in C++, and “printf()” in C

- **System.in**
 - Reads from standard input
 - Equivalent to “cin” in C++, and “scanf()” in C

- **System.err**
 - Prints to standard error
 - Equivalent to “cerr” in C++, and “fprintf(stderr)” in C

System.out.println()

33

```
public static void main(String[] args) {  
    System.out.print("I want to believe that most of you");  
    System.out.println(" want to be a very good programmer.");  
}
```

- Class System supplies objects that can print and read values
- System variable out references the standard printing object
 - Known as the standard output stream
- Variable out provides access to printing methods
 - print(): displays a value
 - println(): displays a value and moves cursor to the next line

Escape sequences

34

- Java provides escape sequences for printing special characters
 - `\b` backspace
 - `\n` newline
 - `\t` tab
 - `\r` carriage return
 - `\\` backslash
 - `\"` double quote
 - `\'` single quote

Escape sequences

35

- What do these statements output?

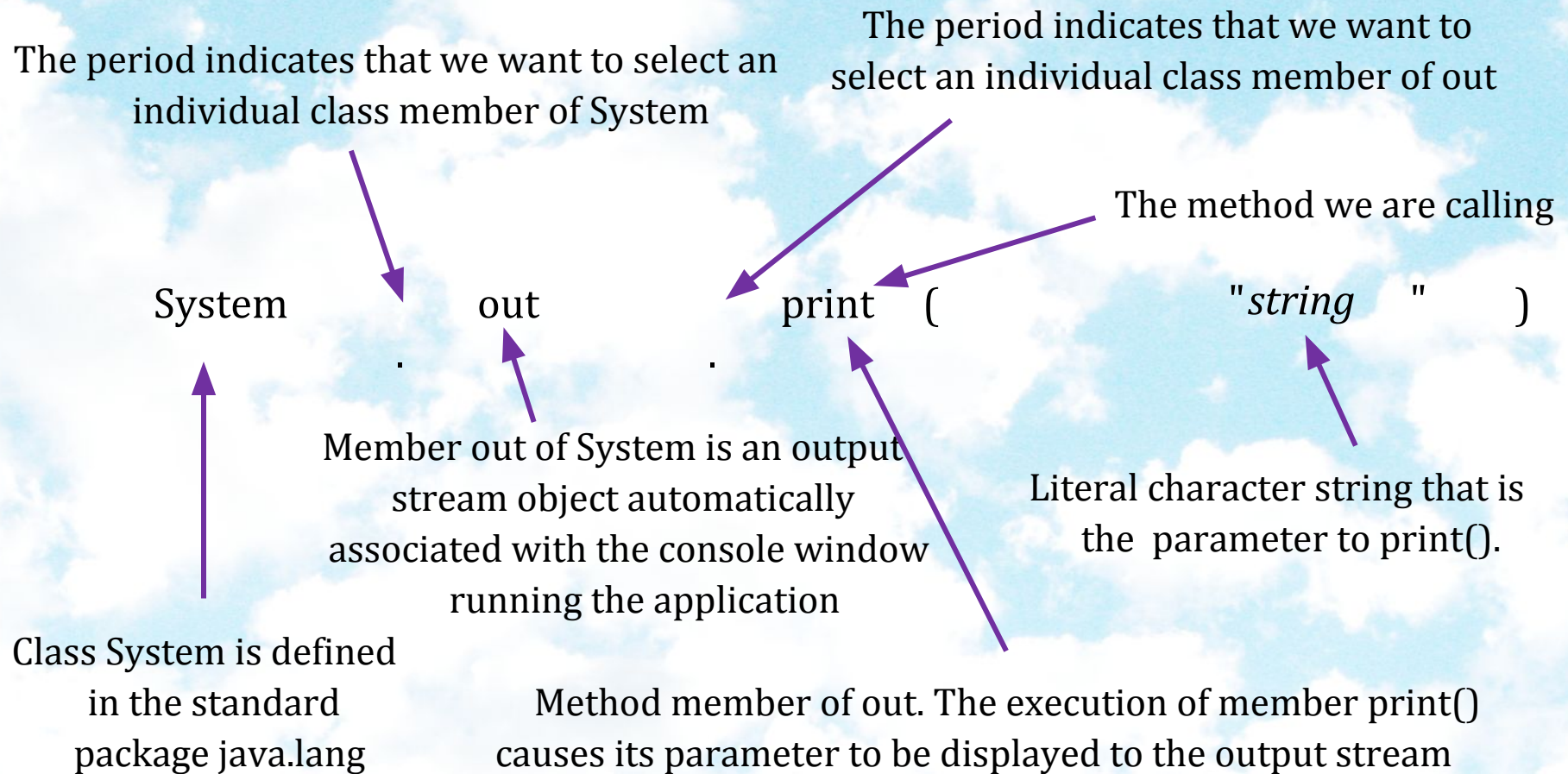
```
System.out.println("Person\tHeight\tShoe size");  
System.out.println("=====  
System.out.println("Hannah\t5'1\"\t7");  
System.out.println("Jenna\t5'10\"\t9");  
System.out.println("JJ\t6'1\"\t14");
```

- Output

```
Person Height Shoe size  
=====  
Hannah 5'1" 7  
Jenna 5'10" 9  
JJ 6'1" 14
```


Selection

36



Example program: temperature conversion

37

```
// Purpose: Convert a Celsius temperature to Fahrenheit
```

```
public class CelsiusToFahrenheit {
```

```
    // main(): application entry point
```

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

```
        // set Celsius temperature of interest
```

```
        int celsius = 28;
```

```
        // convert to Fahrenheit equivalent
```

```
        int fahrenheit = 32 + ((9 * celsius) / 5);
```

```
        // display result
```

```
        System.out.println("Celsius temperature");
```

```
        System.out.println(" " + celsius);
```

```
        System.out.println("equals Fahrenheit temperature");
```

```
        System.out.println(" " + fahrenheit);
```

```
    }
```

```
}
```

Type Conversion & Casting

Type Conversion

39

- ☐ Automatic Type Conversion
- ☐ Casting Incompatible types

Automatic Type Conversion

40

- Automatic Type Conversion:
 - When two types are compatible
 - The destination type is larger than the source type.
 - Example:
 - int type is larger than byte value
 - The numeric types are compatible with each other.
 - The numeric types are not compatible with character or boolean
 - char and boolean are not compatible with each other.

Automatic Type Conversion

41

- ☐ short's variable = byte's variable → ok
- ☐ int's variable = byte's variable → ok
- ☐ byte's variable = int's variable → Error
- ☐ float's variable = int's variable → ok
- ☐ int's variable = float's variable → Error
- ☐ double's variable = float's variable → ok
- ☐ float's variable = double's variable → Error
- ☐ char's variable = any other variable → Error
- ☐ int's variable = char's variable → ok
- ☐ short's variable = char's variable → Error
- ☐ boolean variable = any other variable → Error
- ☐ Any other variable = boolean variable → Error

Casting Incompatible Types

42

- Casting Incompatible Types:
 - When narrowing conversion is occurred.
- Way:
 - (target-type) value
- Example:
 - `int a=20;`
 - `byte b;`
 - `b=(byte) a;`

Casting

43

- Consider the following code

```
double d = 3.6;
int x = Math.round(d);
```
- Java complains (about loss of precision). Why?
- Math.round() returns a long, not an int
 - So this is forcing a long value into an int variable
- How to fix this

```
double d = 3.6;
int x = (int) Math.round(d);
```
- You are telling Java that it is okay to do this
 - This is called “casting”
 - The type name is in parenthesis

More casting examples

44

- Consider

 - `double d = 3.6;`

 - `int x = (int) d;`

- At this point, x holds 3 (not 4!)

 - This truncates the value!

- Consider

 - `int x = 300;`

 - `byte b = (byte) x;`

 - `System.out.println (b);`

- What gets printed?

 - Recall that a byte can hold values -128 to 127
 - 44!
 - This is the “loss of precision”

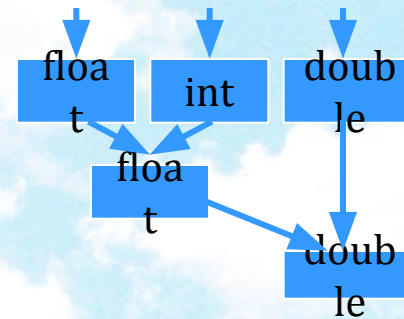
- Java automatically promotes each byte or short operand to int when evaluating an expression.
- Example:
 - `byte a=40, b=50,c=60;`
 - `int d=a*b+c; // here, d will be 2060`
- Problem:
 - `byte b=20;`
 - `b=b*2; //Error: Can't assign an int to a byte`
- Solution:
 - `b=(byte)(b*2);`

The Type Promotion Rules

46

- All byte and short values are promoted to int
- If one operand is a long, the whole expression is promoted to long.
- If one operand is a double, the whole expression is promoted to double.
- How it works:

```
byte b=34;  
char c = 'a';  
short s=1023;  
int i = 343;  
float f=34.46f  
double d = .23  
double result = (f*b) + (i/c) - (d*s);
```



Expressions

47

- What is the value used to initialize expression
`int expression = 4 + 2 * 5;`
- What value is displayed
`System.out.println(5 / 2.0);`
- Java rules in a nutshell
 - Each operator has a precedence level and an associativity
 - Operators with higher precedence are done first
 - * and / have higher precedence than + and -
 - Associativity indicates how to handle ties
 - When floating-point is used the result is floating point

Question on expressions

48

- Does the following statement compute the average of double variables a, b, and c? Why or why not?

`double average = a + b + c / 3.0;`

Using Math Library

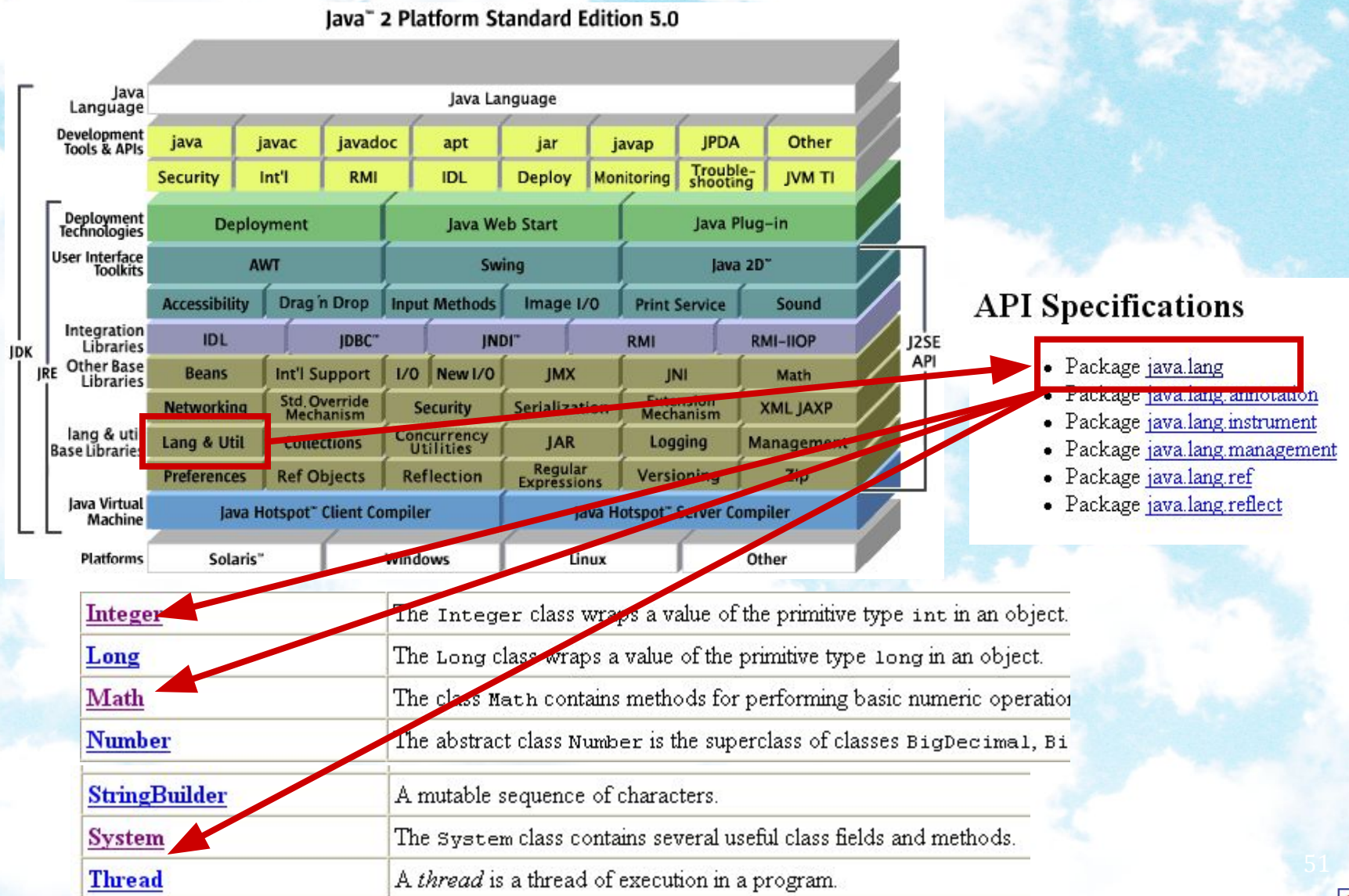
About Math Library

50

- Math class is under the package of java.lang
- The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- How to use Math library:
 - `double a=Math.round(34.64);`
- [Here](#) is some homework.

How to get help (From JDK Documentation)

51



Using Integer Class

52

- Some Math functions: `sin()`, `cos()`, `log()`, `sqrt()`
- Using Integer Object:
 - `int a = Integer.MAX_VALUE;`
 - `int b = Integer.SIZE;`
 - `String str=Integer.toString(123);` `//works as itoa()`
 - `int b=Integer.bitCount(10);`

Reading

53

- Java2: The Complete Reference (Herbert Schildt)
 - Chapter 2: Overview of Java
 - Chapter 3: Data Types, Variables

Homework (Math Library)

54

- Suppose you are given the following
 - `double a=56.34, b=6.58334, c=-34.4265;`
- Calculate the following value:
 - Print the pi's value and e's value
 - Print a random number.
 - Find the absolute value of the variable c
 - Find the square root of a
 - Find the maximum value between a and b
 - Calculate the value a^b
 - Round the number a
 - Calculate the value of $\sqrt{a^2+b^2}$
 - Find the floor, ceil and round value of b and c
 - Find the radian value of a.
 - Find the sin value of a where a represents the degree

Thank you