EE3206 Java Programming and Applications

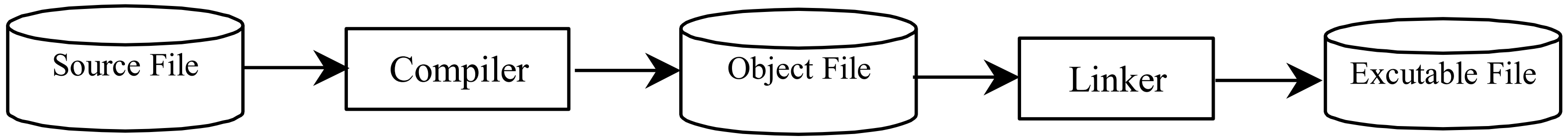
Lecture 1 Introduction to Java

Intended Learning Outcomes

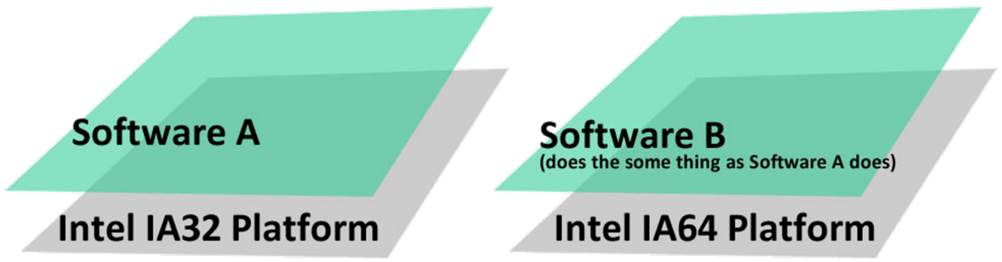
- To understand the Java runtime environment.
- To know Java's terminologies, advantages and characteristics.
- To recognize the form of identifiers.
- To learn Java primitive data types and object types.
- To become familiar with Java programming style and naming conventions.
- To create, compile, and run a simple Java program.
- To understand method overloading.
- To determine the scope of local variables.
- To understand Array of Java.
- To recognize the multidimensional form of arrays.
- To learn common String operations in Java.
- To use the Character class to process a single character.
- To use the StringBuffer class to process flexible strings.
- To learn how to pass strings to the main method from the command line.

Platform Dependent Language

- Program called a compiler is used to translate the source program into a machine language program called an object program. The object program is often then linked with other supporting library code before the object can be executed on the machine.

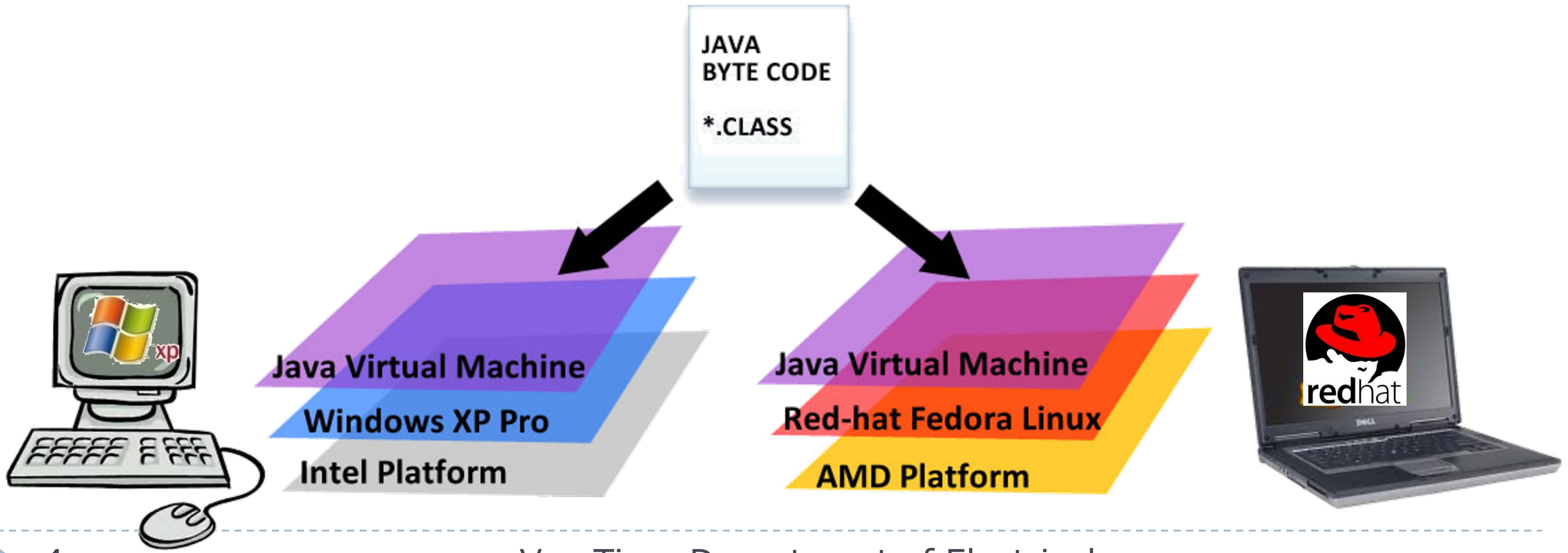


- Platform Dependent Language (e.g. C, C++)
- You can port a source program to any machine with appropriate compilers that convert the source code to a specific native machine code. The source program must be recompiled, however, because the object program can only run on a specific machine.

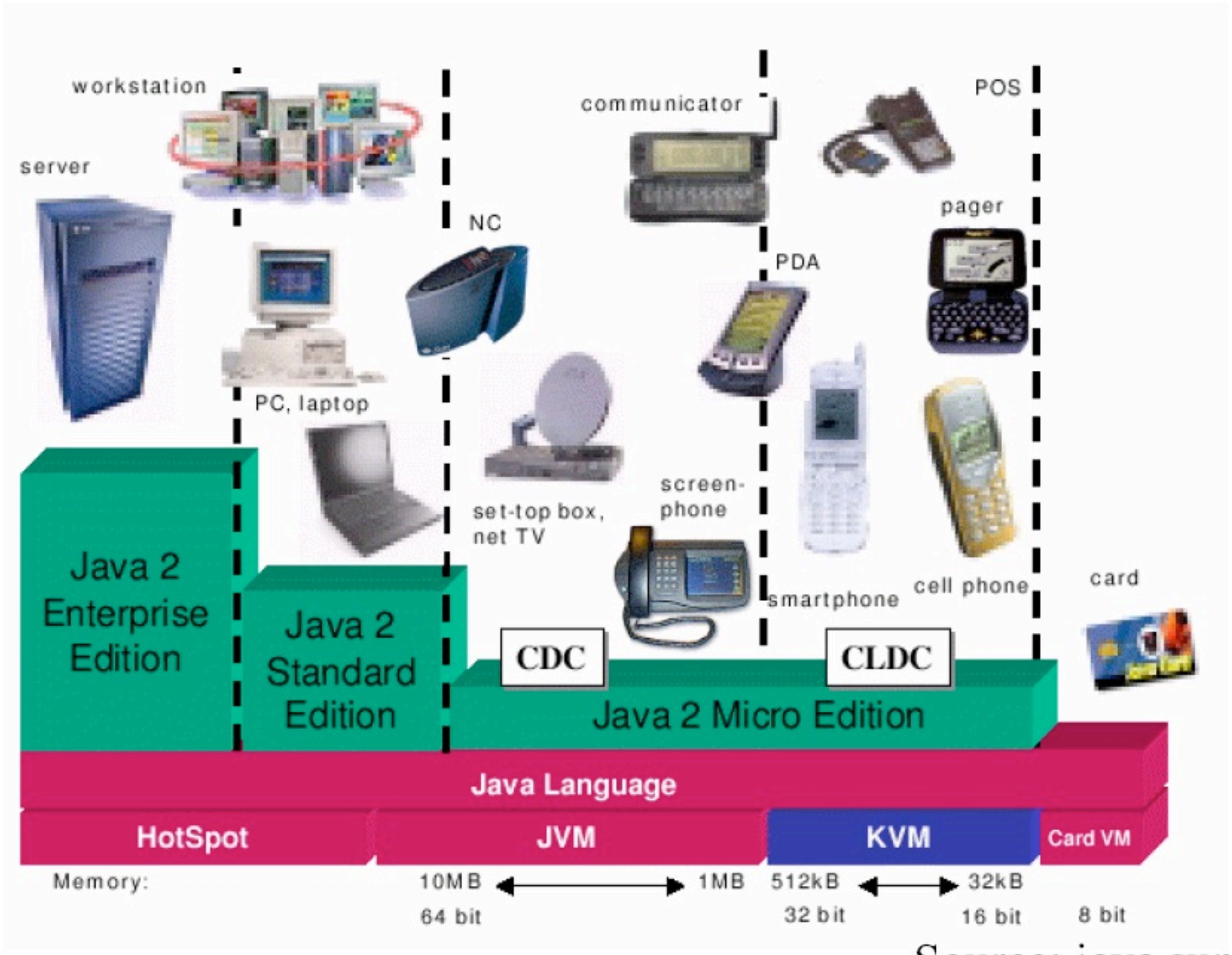


Platform Independent Language

- Platform Independent Language (e.g. Java)
- Nowadays computers are networked to work together. Java was designed to run on any platform. You write the program once, and compile the source program into a special type of object code, known as bytecode. The bytecode can then run on any computer with a Java Virtual Machine (JVM) which is a software interpreter for Java bytecode.



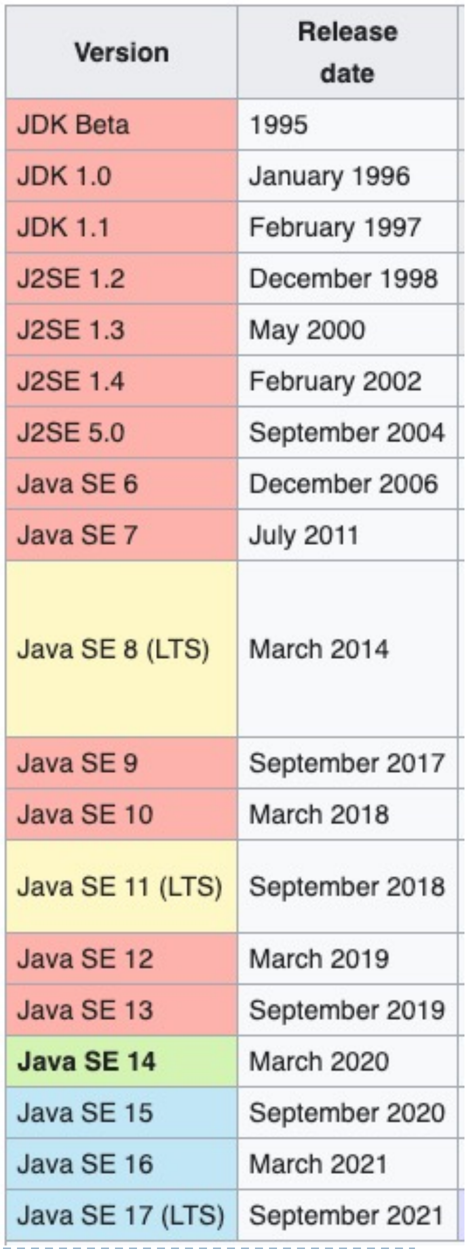
Java Platforms & Editions



Source: java.sun.com

JDK New License Model

- Since JDK 9, Oracle releases a new version of JDK every 6 months (on Mar and Sep). Before that, it was normally between 2 to 5 years for each new release.
- Each release has a lifespan about 6 months until the next release and will not receive further updates except Long-Term-Support (LTS) version.
  - Current LTS release is JDK8 and JDK 11 (*and JDK 17 on Sep 2021*)
- JDK vs OpenJDK
  - From Java 11 forward, they are fundamentally the same in terms of features. All features available in JDK build is also available in OpenJDK build.
- JDK is not free for commercial use. Oracle provides LTS.
- OpenJDK is free for commercial use, but receives no LTS. Company needs to upgrade to the next release every 6 months.



Version	Release date
JDK Beta	1995
JDK 1.0	January 1996
JDK 1.1	February 1997
J2SE 1.2	December 1998
J2SE 1.3	May 2000
J2SE 1.4	February 2002
J2SE 5.0	September 2004
Java SE 6	December 2006
Java SE 7	July 2011
Java SE 8 (LTS)	March 2014
Java SE 9	September 2017
Java SE 10	March 2018
Java SE 11 (LTS)	September 2018
Java SE 12	March 2019
Java SE 13	September 2019
Java SE 14	March 2020
Java SE 15	September 2020
Java SE 16	March 2021
Java SE 17 (LTS)	September 2021

Why Java?

- Java is:
  - Simple and Object-Oriented (simplify from C++)
  - Distributed (support network programming)
  - Interpreted, portable and architecture-neutral
  - Robust and secure (exception handling, digital signature)
  - High-performance (hotspot technology)
    - Although it is suffered from the performance degradation of bytecode interpretation, some analysis show that Java's performance is quite close to native code nowadays
  - Multi-threaded (simultaneous tasks)
  - Dynamic (no installation, no recompilation, load-on-the-fly)

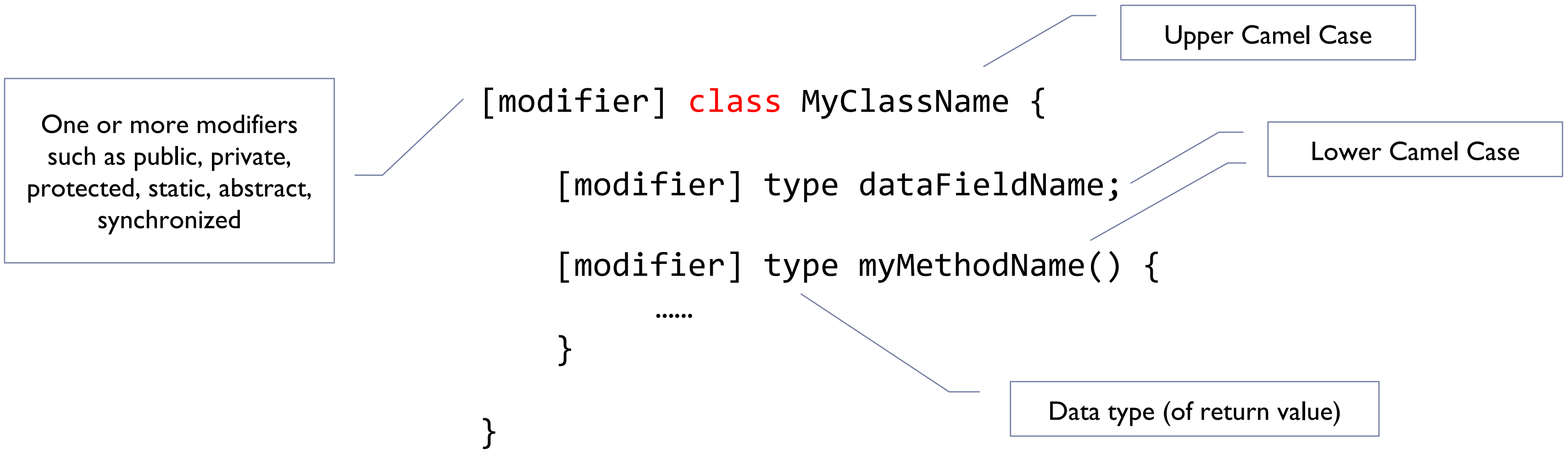
Hello Java!

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

- This is a simple program that prints out the string “Hello Java!” to console.
- Unlike C++, Java function must be written inside a class. Every Java program has at least one class that serve as a placeholder for the main function.

Java is OO Language

- Object-Orientation encourages the use of objects to model or represent your program.
- A class is the blueprint/definition of objects.
- Data field and method are defined in a class.



```
[modifier] class MyClassName {

    [modifier] type dataFieldName;

    [modifier] type myMethodName() {
            ……
    }

}
```

Upper Camel Case
One or more modifiers such as public, private, protected, static, abstract, synchronized
Lower Camel Case
Data type (of return value)

Class and Object

- An object is a collection of variables and functions (similar to struct of C but more powerful).
- Objects can exchange data and invoke functions of each other. This capability constitutes the business logic of a program.
- In an OO program, objects are essentially everything. A running Java program is considered as the interactions between a group of objects.

- A class is a template or blueprint for building objects.
- A class is a Java construct where you can specify the features (e.g. variables and functions) of the desired object to be created from this class.

Method and Modifier

- A method is a collection of statements that performs a sequence of operations. It is equivalent to what a "function" means in C.
- A method must be defined together with a class.

- A modifier is used to specify the properties of the data, methods, and classes.
- Some common modifiers are public, protected, private, final, static and abstract.

Package

- A Java package is a mechanism for organizing Java classes into different namespaces.
- Programmers typically use packages to organize classes belonging to the same category or providing similar functionality.
- Logical package structure is mirrored to the physical file path:
  - java.lang.Math (class Math in package java.lang)
  - $CLASSPATH/java/lang/Math.class

main Method

- Any programs must provide an entry point for the start of execution. The Java interpreter executes the application by invoking the main method.
- The main method usually provides the control of program flow.
- The main method has an argument in the type of String array and no return value:

  ```
  public static void main(String[ ] args) {
      // Statements;
  }
  ```

Steps to Write a Java Program

1. Think about what types of objects you need in your program.
2. Define a class for each type of the desired objects. This will also define the data structure and operations of the type of objects.
3. Define a Main class that acts as a placeholder for the main method. The main method can be embedded in any classes in your Java program.
4. Initialize your objects in the main method and initiate any business operations.

See example

Example - Hello GPA

```java
class Main {
    // This is the entry point of a program
    public static void main(String[] args) {

        // create two objects of Student type
        Student bill = new Student("Bill Chan", 1.5);
        Student larry = new Student("Larry Chow", 4);

        // display their academic standing
        bill.showAcademicStanding();
        larry.showAcademicStanding();
    }
}
```

```java
class Student {

    // Declare the variables you need to represent a student
    String name = "";
    double gpa = 0;

    /*  Used to configure the values of a student object */
    Student(String studentName, double studentGpa) {
        name = studentName;
        gpa = studentGpa;
    }

    /** Display the academic standing of a student */
    void showAcademicStanding() {
        System.out.print("The Academic Standing of " + name + " is: ");
        if(gpa > 1.7)
            System.out.println("Good Standing");
        else
            System.out.println("Academic Warning");
    }
}
```

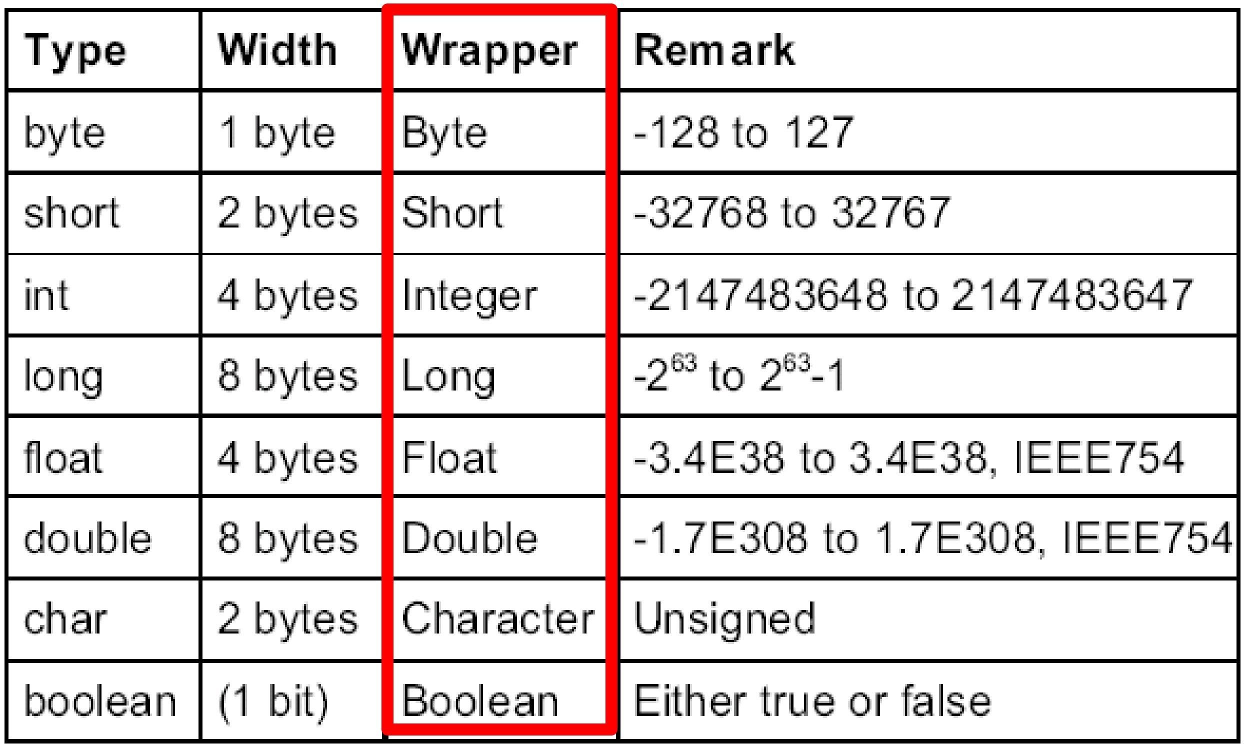
Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier cannot start with a digit.
- An identifier cannot be a reserved word.
- An identifier cannot be true, false or null.
- An identifier can be of any length.

- Legal identifiers
  - _2numberOfRow, $person, anyNumberOfLetter
- Illegal identifiers
  - 2numberOfRow, public

8 Primitive Data Types

- Each primitive type has a wrapper type that not only stores value but also provides operations on the data unit.
- The wrapper is an OO counterpart for the primitive data type.



Type	Width	Wrapper	Remark
byte	1 byte	Byte	-128 to 127
short	2 bytes	Short	-32768 to 32767
int	4 bytes	Integer	-2147483648 to 2147483647
long	8 bytes	Long	-2^{63} to $2^{63}-1$
float	4 bytes	Float	-3.4E38 to 3.4E38, IEEE754
double	8 bytes	Double	-1.7E308 to 1.7E308, IEEE754
char	2 bytes	Character	Unsigned
boolean	(1 bit)	Boolean	Either true or false

Declare and Initialize Variables

- Declaration:
  - datatype varName;
- Assignment:
  - varName = value;
- Initialization:
  - datatype varName = value;

- Examples:
  - int x; // declare x to be an integer variable
  - x = 1; // assign 1 to x;
  - double d = 1.4; // initialize d to 1.4
  - char ch1, ch2; // declare ch1 and ch2 as a character

Constants

- General Format:
  - final datatype CONSTANTNAME = VALUE;
- Example:
  - final double PI = 3.14159;
  - final int SIZE = 3;
- The keyword final can also be used to modify a class or method. This will be discussed later.

Literals

- A literal is a constant value that appears directly in the program source.
  - int x = 34;   // 34 is referred as a literal
- By default, a fractional number is treated as a double type value. For example, 5.0 is considered a double value, not a float value.
  - double y = 5.0;
- As a result, the following statement is wrong
  - float z = 5.0;   // assign a double value to a float variable
- To make a number a float, append the letter F at the end (either in lowercase or uppercase)
  - float z = 5.0f;   // this is correct
- Use the following letter to denote other data types:
  - D – double
  - L – long
- Double literals can also be specified in scientific notation with E as an exponent.:
  - double s = 1.2345E-2;   // 0.012345

Type Widening and Narrowing

- When performing a binary operation involving two operands of different types, Java automatically converts/promotes the operand of smaller range to the type of larger range.
- This is done automatically and hence it is called implicit casting.

```
byte b = 10;
int i = b + 1000;           // byte to int
double d = 1000;            // int to double, lossy
```

- Sometimes operands are converted from larger range to a type of smaller range. This leads to truncation of some least significant bits and errors.
- Programmers have to force casting of variables explicitly, hence it is called explicit casting.

```
int i = (int)3.0;           // type narrowing
int i = (int)3.9;           // fraction part is truncated
float f = 2.3;              // error, why?
int x = 5 / 2.0;            // error, why?
```

The String Type

- In contrast to C where string is represented by a character array, Java uses the type String to store a string.
- The String type is not a primitive type. It is known as a reference type or object type.
- String can be concatenated with other types
  - // "Welcome" is a string literal
  - String message = "Welcome";

  - // Three strings are concatenated
  - String message = "Welcome " + "to " + "Java";

  - // String Chapter is concatenated with number 2
  - String s = "Chapter" + 2; // s becomes Chapter2

  - // String Supplement is concatenated with character B
  - String s = "Supplement" + 'B'; // s becomes SupplementB

Programming Style and Documentation

- Programming styles and conventions are often overlooked by beginners. Not obeying conventions does not harm the correctness of your code. However, they are important for keeping a program structural, readable and maintainable.
- They are the keys for being a successful programmer.

  - Appropriate Comments
  - Naming Conventions
  - Proper Indentation and Spacing Lines
  - Block Layout

Comments

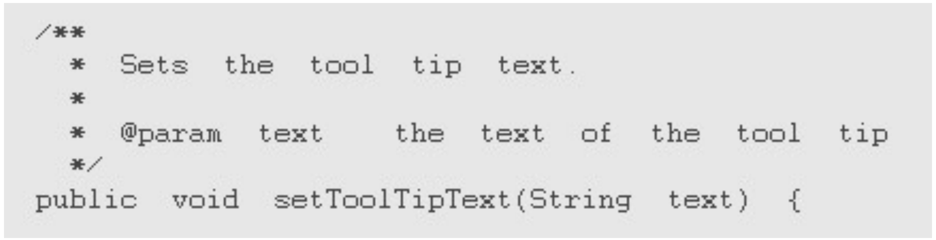
- In Java, comments are
- preceded by two slashes (//) in a line; or
  - // single line comment …

- enclosed between /* and */ in one or multiple lines; or
  - /* multiple
  -       lines
  - */

- enclosed between
  - /** multiple
  -        lines
  - */
  - A special format used by javadoc tool to compile into API specification.
    - A webpage-like HTML file

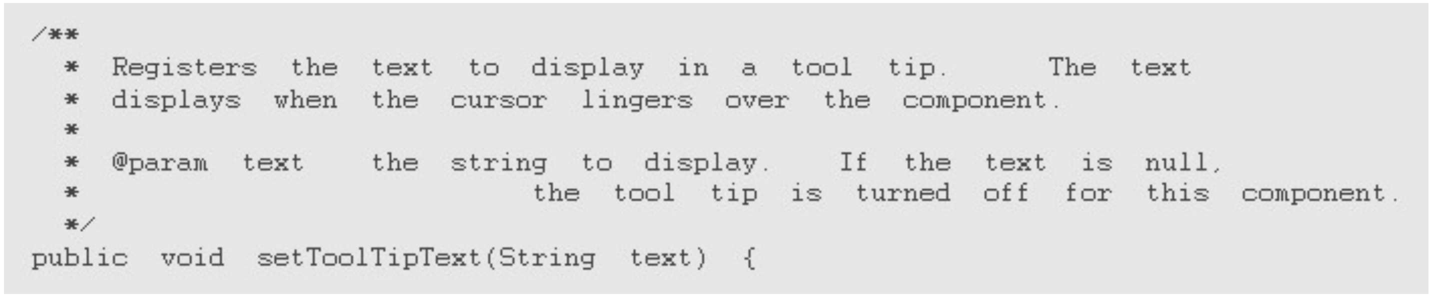
Appropriate Comments

- Include a summary at the beginning of the class or method to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
- For our coursework, include your name, student ID, class section, date and a brief description at the beginning of the program.



```
/**
  * Sets the tool tip text.
  *
  * @param text  the text of the tool tip
  */
public void setToolTipText(String text) {
```

Which one is better?



```
/**
  * Registers the text to display in a tool tip.   The text
  * displays when the cursor lingers over the component.
  *
  * @param text  the string to display.  If the text is null,
  *                   the tool tip is turned off for this component.
  */
public void setToolTipText(String text) {
```

Naming Conventions

- Java usually does not use abbreviation for naming, which often becomes a source of confusion in C language. You should choose meaningful and descriptive names.
  - fn → fileName
  - ecode → errorCode

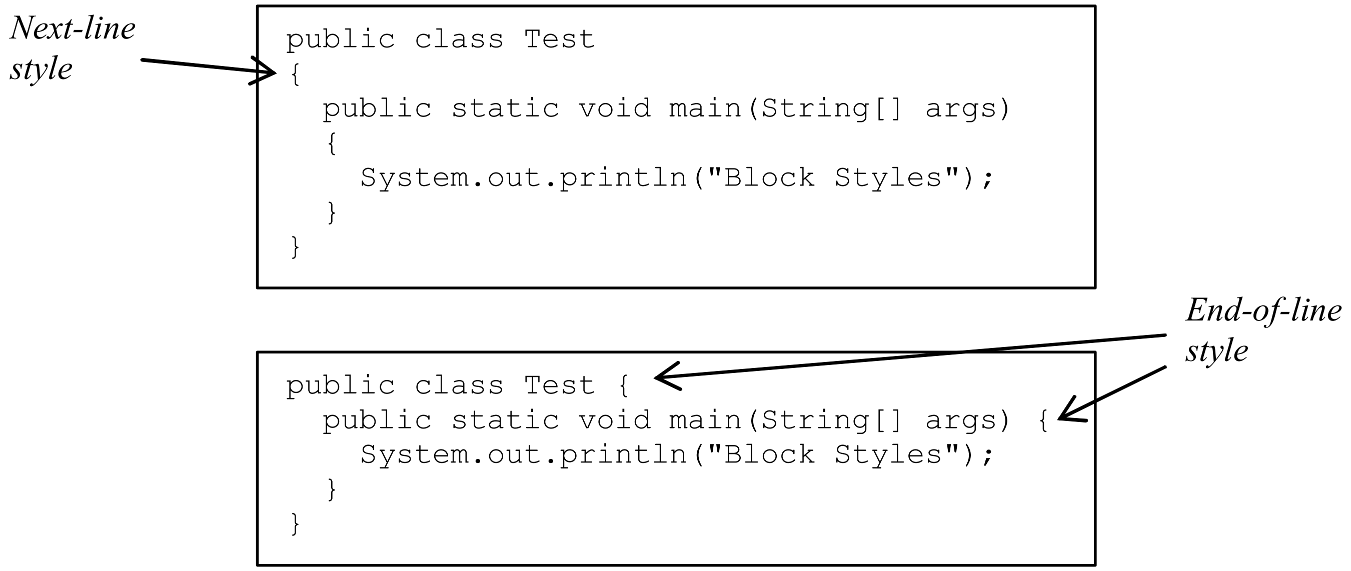
- Variable and Method names (Lower Camel Case)
  - Start with lowercase and capitalize every first letter of each subsequent word.
  - radius, outerArea, computeArea, getLocalPoint.

- Class names (Upper Camel Case)
  - Capitalize the first letter of each word in the name.
  - GameScreen, ControllerButton

- Constants
  - Capitalize all letters in constants, and use underscores to connect words.
  - PI, MAX_VALUE

Proper Indentation, Spacing and Block Layout

- Indentation
  - Indent two spaces.
- Spacing
  - Use blank line to separate segments of the code.
- Block Layout
  - Use end-of-line style for braces.



```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

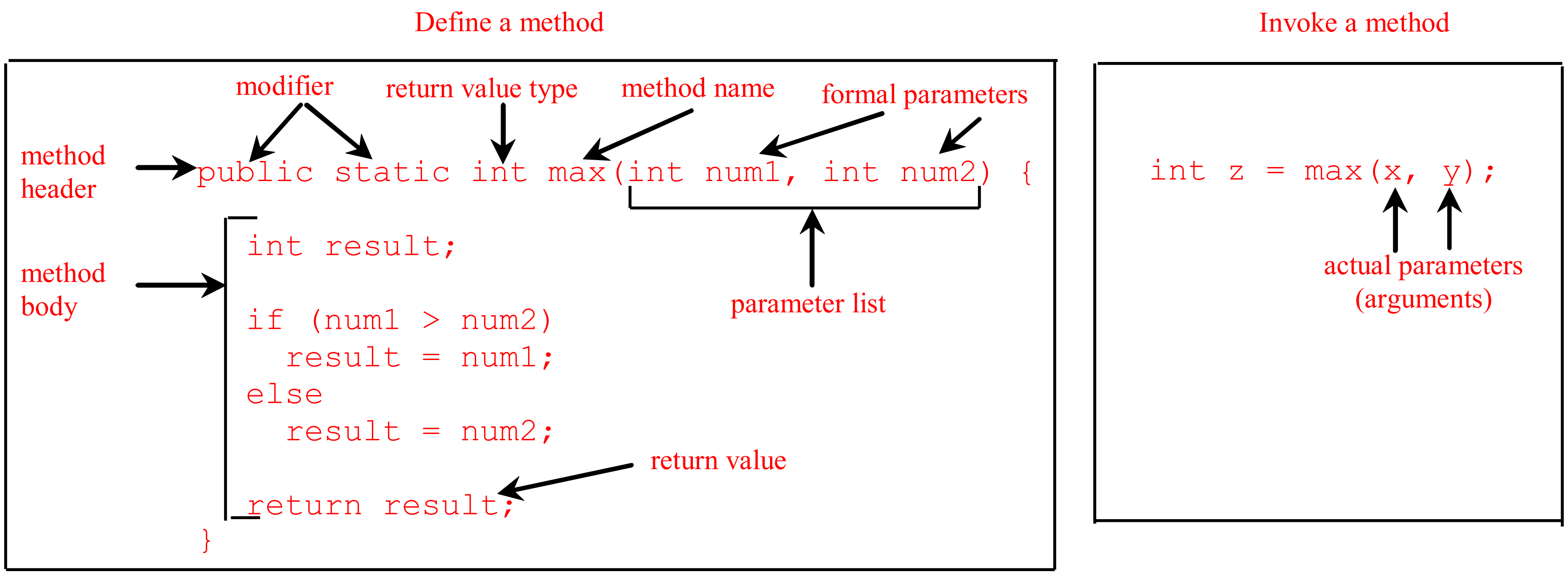
```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

Control Statements

- There are a few control statements same as that of C/C++ language. They are used to control the flow of a program.
  - if, if-else
  - switch
  - for-loop, while-loop, do-while-loop
  - break, continue

What is Method?

- A method is:
  - a collection of statements that are grouped together to perform an operation.
  - an execution routine or a building block.
  - the counterpart of the function in C language



```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

```
int z = max(x, y);
```

Passing Arguments

- In Java, there are two ways to pass an argument
  - All primitive data are passed by VALUE.
    - A copy of the actual parameters is passed to the method
    - Modifying the formal parameters inside the method DOES NOT affect the actual parameters
  - Objects are passed by REFERENCE.
    - The virtual address (reference) of the actual parameters is passed to the method
    - Modifying the formal parameters inside the method DOES affect the actual parameters
    - Because of this property, object type is also known as reference type

Overloading Methods

- Method signature is the combination of the method name and the parameter list.
  - Method signature = method name + parameter lists

- Java compiler identifies the method being invoked by the method signature.
- Two methods can be declared with different signatures, hence using same name but different parameter lists is allowed.
- This is called method overloading.

TestMethodOverloading

Ambiguous Invocation

- Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match. This is referred to as ambiguous invocation. Ambiguous invocation is a compilation error.

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

```
public static void main(String[] args) {
   Ambiguous.max(1, 2);
 }
```

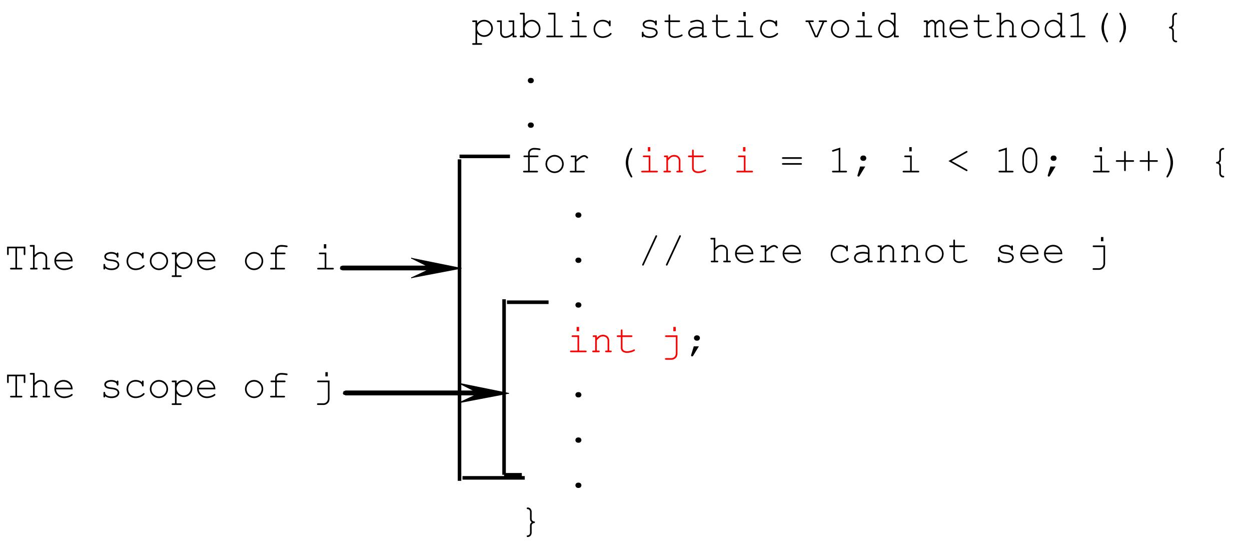
Scope of Variables

- Data field (of class)
  - A variable defined inside a class but outside a method.
- Local variable (of method)
  - A variable defined inside a method.

- Scope
  - Refer to the part of the program where the variable can be "seen" or referenced.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
- A local variable must be declared before it can be used, but cannot be re-declared in the same scope.

Scope of Local Variables

- A variable declared in the initial action part of a for loop header has its scope in the entire loop.
- But a variable declared inside a for loop body has its scope limited in the loop body from its declaration to the end of the block that contains the variable.



```
public static void method1() {
  .
  .
  for (int i = 1; i < 10; i++) {
    .
    .  // here cannot see j
    .
    int j;
    .
    .
    .
  }
```

Scope of Local Variables

It is fine to declare i in two non-nesting blocks

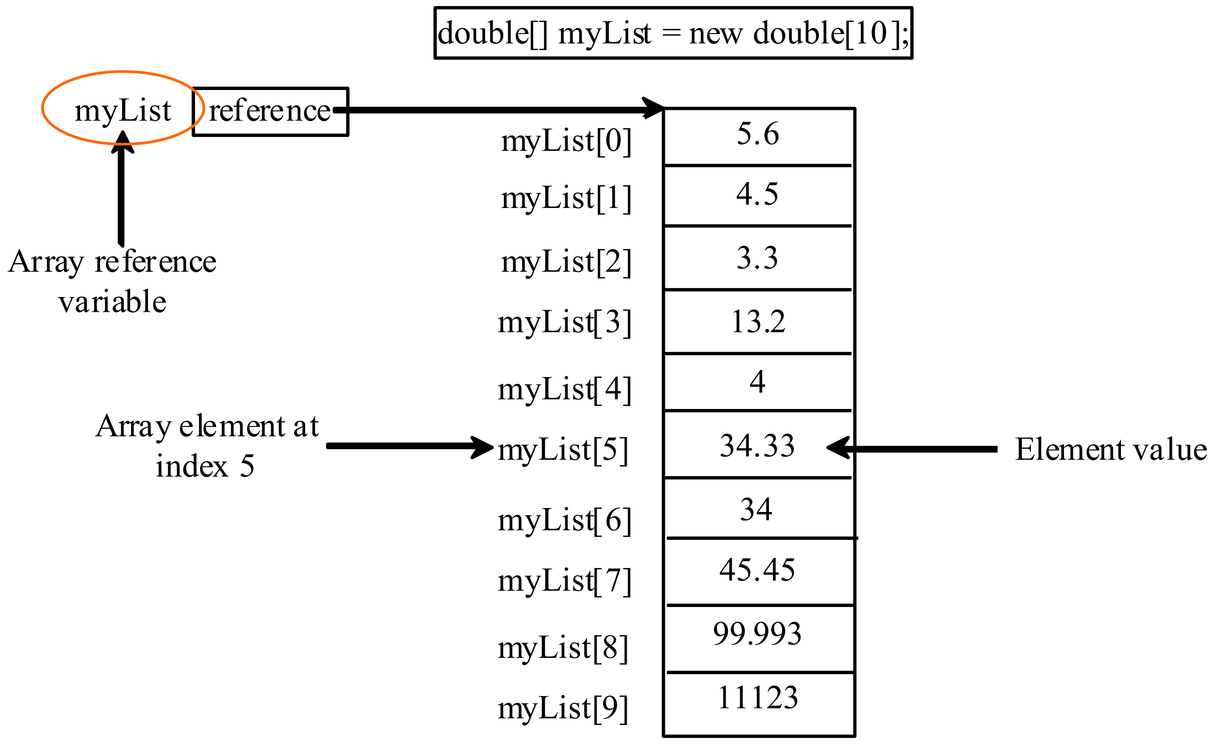
```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

It is wrong to declare i in two nesting blocks

```
  public static void method2() {

    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++)
      sum += i;
    }

  }
```

Arrays

- Array is a data structure that represents a collection of the same types of data.
- Java Array itself is an object which is a reference type, no matter what type of elements it contains.



Declaring & Creating Array

- To declare an array
  - datatype[] arrayRefVar;
  - Example: double[] myList;

- To create an array
  - arrayRefVar = new datatype[arraySize];
  - Example: myList = new double[10];

- To declare & create in one step
  - datatype[] arrayRefVar = new datatype[arraySize];
  - Example: double[] myList = new double[10];

Array On Creation

- An array’s size is fixed once created
  - arrayRefVar.length
- Default value on creation
  - 0 for the numeric primitive data types,
  - '\u0000' for char types, and
  - false for boolean types.
- Array’s elements accessed through index
  - array indices are 0-based
  - indexed variable format: arrayRefVar[index];
  - e.g. myList[2] = myList[0] + myList[1];

Initializing Array Using Shorthand

- The shorthand notation must be in one statement.
  - double[] myList = {1.9, 2.9, 3.4, 3.5}; // correct
  - double[] myList; // only declare
  - myList = {1.9, 2.9, 3.4, 3.5}; // wrong

- The above shorthand notation is equivalent to the following statements:
  - double[] myList = new double[4];
  - myList[0] = 1.9;
  - myList[1] = 2.9;
  - myList[2] = 3.4;
  - myList[3] = 3.5;

Example - Number Counter

- Objective: The program receives 6 numbers from the user, finds the largest number and counts the occurrence of the largest number entered.

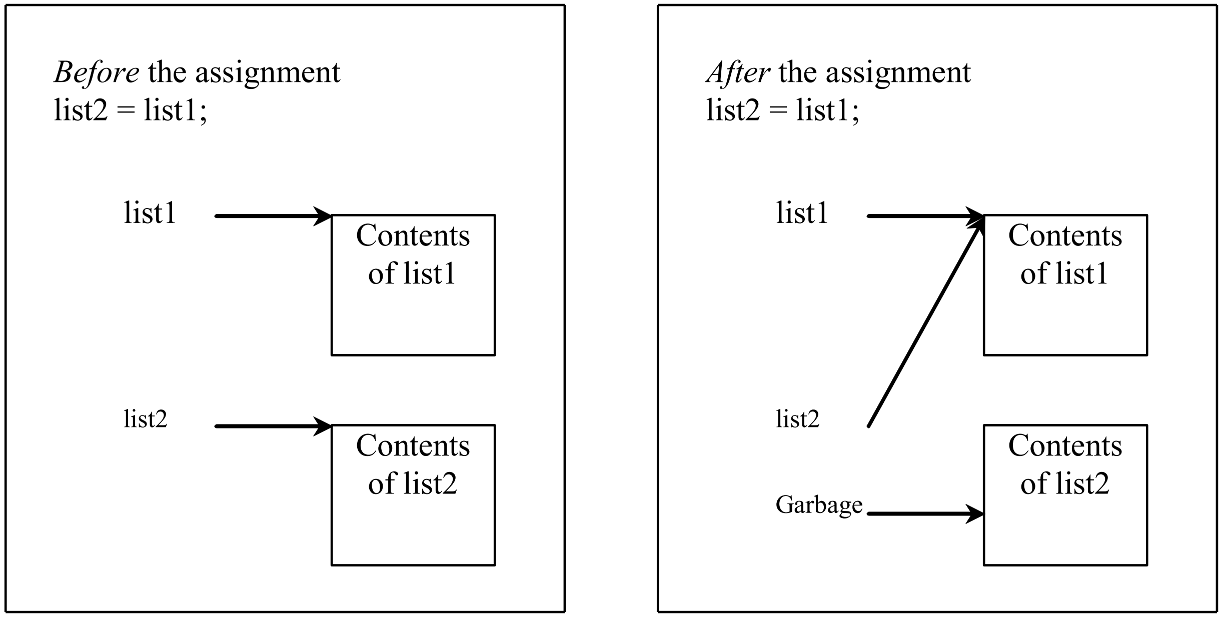
- Suppose you entered 3, 5, 2, 5, 5, and 5, the largest number is 5 and its occurrence count is 4.

  - Note 1: the keyword "this" is omitted in this example
  - Note 2: the dummy class for main method is skipped

NumberCounter

Copying Arrays (Shallow Copy)

- Often, you need to duplicate an array or a part of an array. If you attempt to use the assignment statement (=), as follows:
  - list2 = list1;



- Now list2 references to the contents of list1.
- This is <u>not a real copy</u>. It is also known as Shallow Copy.

Copying Arrays (Deep Copy)

- In contrast to shallow copy, deep copy duplicates all the elements in an array.

- Using a loop:

```
    int[] sourceArray = {2, 3, 1, 5, 10};
    int[] targetArray = new int[sourceArray.length];

    for (int i = 0; i < sourceArrays.length; i++)
            targetArray[i] = sourceArray[i];
```

- Using arraycopy utility from System Class:
  - arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);

  - Example:
  - System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);

Memory Heap

- Objects and arrays are big in size. Moving them takes many I/O operations that is not preferred and mostly not necessary.
- The JVM stores objects and arrays in an area of memory, called heap, which is used for dynamic memory allocation where blocks of memory are allocated and released in an arbitrary order.
- In source code, the **new** operator is used to allocate memory for the specific type of object.

Two-dimensional Arrays

- // Declare array ref var
- dataType[][] refVar;

- // Create array and assign its reference to variable
- refVar = new dataType[10][10];

- // Combine declaration and creation in one statement
- dataType[][] refVar = new dataType[10][10];

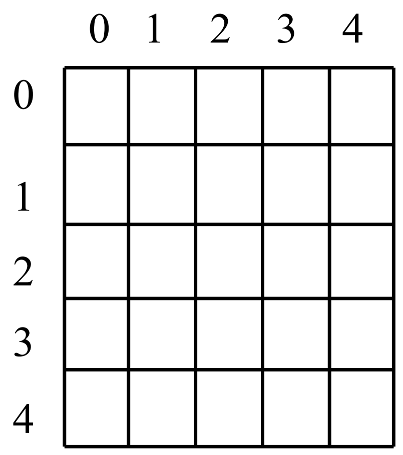
- You can also use an array initializer to initialize a two-dimensional array.

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

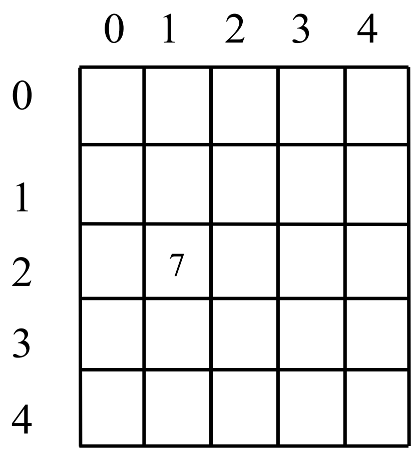
Same as

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2;
array[0][2] = 3; array[1][0] = 4;
array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8;
array[2][2] = 9; array[3][0] = 10;
array[3][1] = 11; array[3][2] = 12;
```

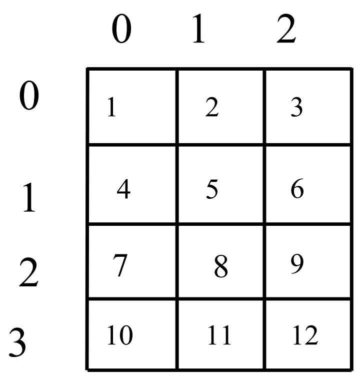
Two-dimensional Array Illustration



```
matrix = new int[5][5];
```



```
matrix[2][1] = 7;
```



```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

matrix.length? 5

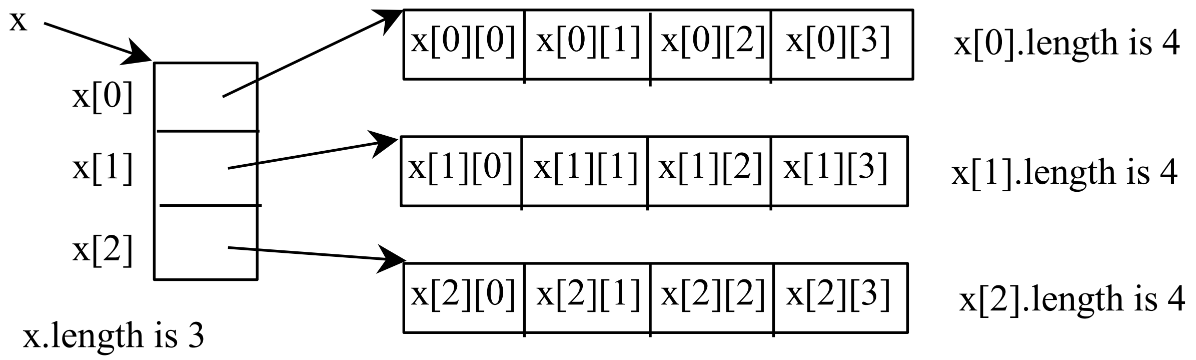
matrix[0].length? 5

array.length? 4

array[0].length? 3

Lengths of Two-dimensional Arrays

- 2D array has a two-stage reference
- int[][] x = new int[3][4];



```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

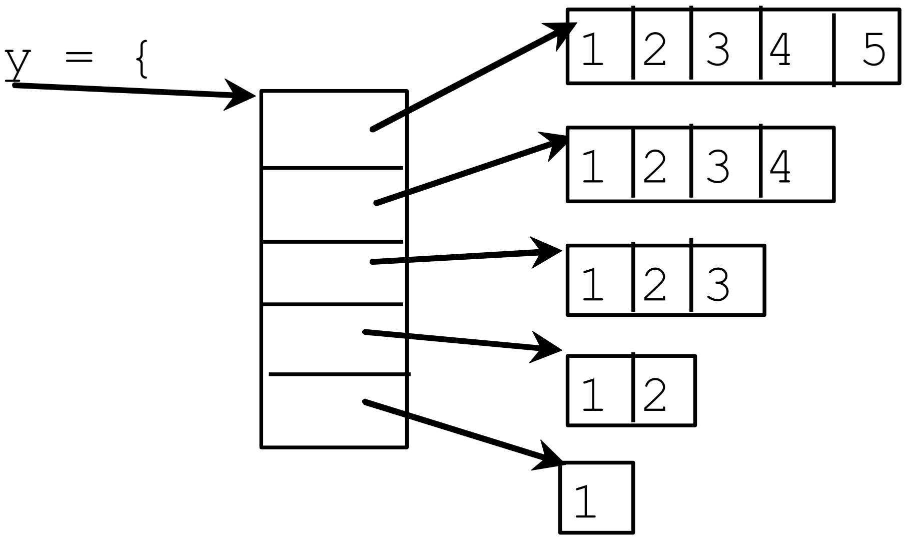
array.length
array[0].length
array[1].length
array[2].length
array[3].length

To access array[4].length, you will get ArrayIndexOutOfBoundsException.

Ragged Arrays

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as a ragged array. For example,

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {1, 2, 3, 4},
  {1, 2, 3},
  {1, 2},
  {1}
};
```



Multidimensional Arrays

- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.

- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays.

- For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

  - double[][][] scores = new double[10][5][2];

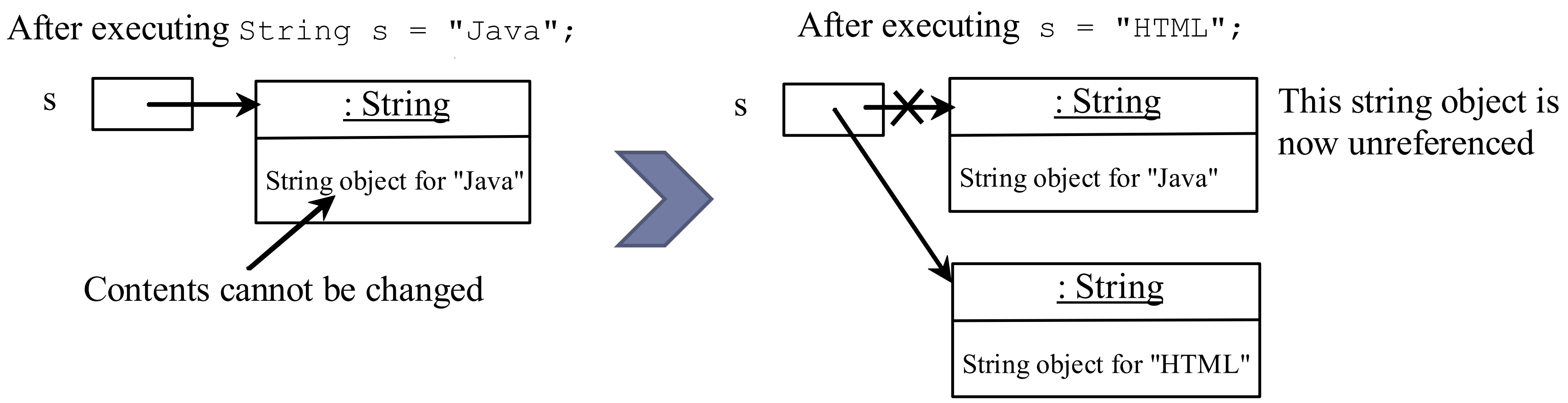
The String Class

- Common string operations
  - Obtaining string length
  - Retrieving individual characters in a string
  - String concatenation
  - Extracting substrings
  - String comparisons
  - String conversions
  - Searching a character or a substring in a string
  - Conversions between strings and arrays
  - Converting characters and numeric values to strings

Constructing Strings

- To construct a String:
  - String message = “Welcome to Java”; // shorthand initializer
  - String message = new String(“Welcome to Java”); // formal constructor
  - String s = new String();

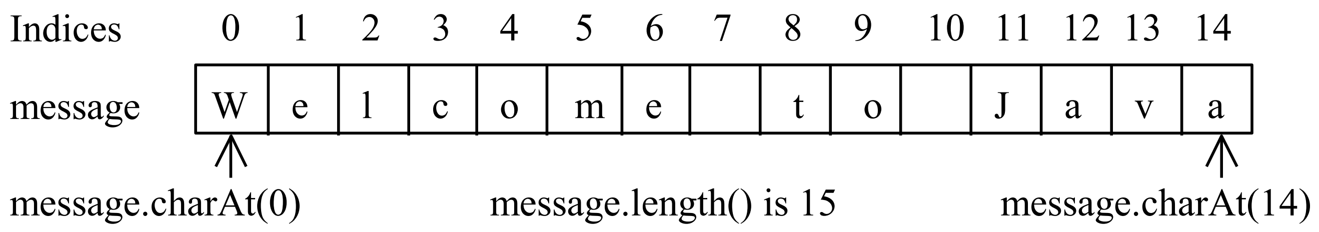
- A String object is immutable, which means its contents cannot be changed once created. For example, the following code does not change the original String object.
  - String s = "Java";
  - s = "HTML";


After executing `String s = "Java";`
s
: String
String object for "Java"
Contents cannot be changed
After executing `s = "HTML";`
s
: String
String object for "Java"
This string object is now unreferenced
: String
String object for "HTML"

Finding String Length and Characters

- Finding string length using the length() method:
  - String message = "Welcome";
  - message.length() (returns 7)

- Using charAt(int) method to retrieve individual character.
  - Do not use message[0]
  - Use message.charAt(index)
  - Index starts from 0



Concatenating Strings

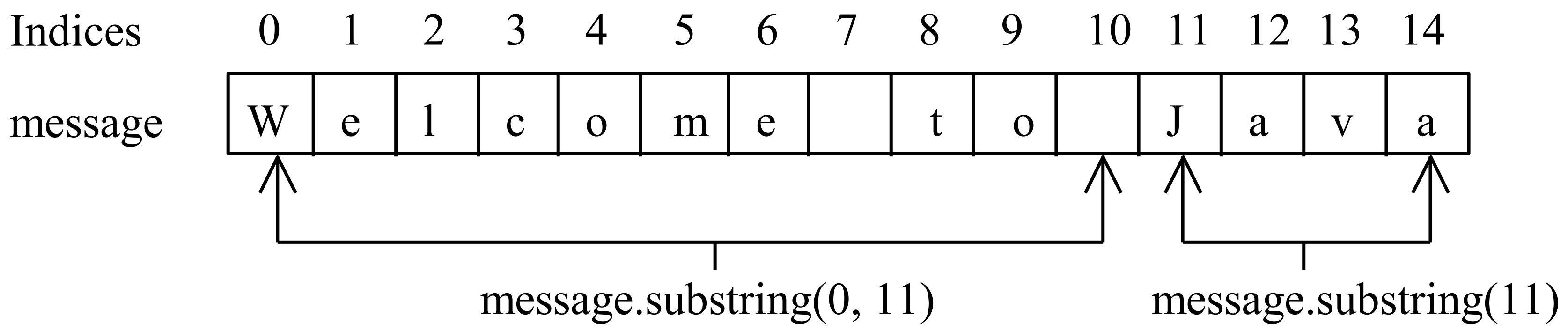
- Use concat() method or + operator to connect two strings:

- Example:
  - String s3 = s1.concat(s2);
  - *is equivalent to*
  - String s3 = s1 + s2;

  - String s0 = s1 + s2 + s3 + s4 + s5;
  - *is equivalent to*
  - String s0 = (((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

Extracting Substrings

- You can extract a single character from a string using the charAt method. You can also extract a substring from a string using the substring method in the String class.
- substring(start) returns a string from (start) to the end of the string.
- substring(start, end) returns a string from (start) to (end - 1).
  - String s1 = "Welcome to Java";
  - String s2 = s1.substring(0, 11) + "HTML"; // becomes "Welcome to HTML"



Comparing Strings

- Use equals() method instead of == operator to compare two strings.
  - return true if equal, otherwise false.
- You can also use compareTo() method to compare two Strings in lexicographical order.
  - return 0 for equal, >0 for greater, and <0 for lesser

```
String s1 = new String("Welcome");
String s2 = new String("Welcome");

 if (s1.equals(s2)){            // true
   // s1 and s2 have the same contents
 }
 if (s1 == s2) {                // false
   // s1 and s2 have the same reference
 }
```

```
String s1 = "Welcome";    // ascii of 'W' is 87
String s2 = "welcome";    // ascii of 'w' is 119
String s3 = "wElcome";    // ascii of 'E' is 69, 'e' is 101
String s4 = "Welcomes";
String s5 = "Welcome";

System.out.println(s1.compareTo(s2));  // (W-w) returns -32
System.out.println(s2.compareTo(s3));  // (e-E) returns 32
System.out.println(s1.compareTo(s4));  // returns -1
System.out.println(s4.compareTo(s1));  // returns 1
System.out.println(s5.compareTo(s1));  // returns 0
```

Converting Strings

- The contents of a string cannot be changed once the string is created. But you can convert a string to a new string using the following methods.
- The methods return a modified new string while the original string remains unchange.
  - toLowerCase() // change all letters to lower case
  - toUpperCase()
  - trim() // remove leading and trailing spaces
  - replace(oldChar, newChar) // replace all oldChar with newChar in the string

Example:

```
String s1 = "Hello World";
String s2 = s1.toLowerCase();
System.out.println(s1 + ' ' + s2);     // Hello World hello world
```

Searching a Character or a Substring

- To search forward, use indexOf()
- To search backward, use lastIndexOf()
- These methods return the position of the first occurrence of the search target

```
String s = “Welcome to Java”;
s.indexOf('W');            //returns 0.
s.indexOf('x');            //returns -1
s.indexOf('o', 5);         //returns 9.
s.indexOf("come");         //returns 3.
s.indexOf("Java", 5);      //returns 11.
s.indexOf("java", 5);      //returns -1.
s.lastIndexOf('a');        //returns 14.
```

Search starts from index 5

Converting Other Types to Strings

- The String class provides several static valueOf() methods for converting a character, an array of characters, and numeric values to strings.
- These methods have the same name valueOf with different argument types char, char[], double, long, int, and float.
- For example, to convert a double value to a string, use String.valueOf(5.44). The return value is a string representation of "5.44".

The Character Class

- To construct a Character object:
  - Character charObject = new Character('b');

- To compare two Characters:
  - charObject.compareTo(new Character('a')); //returns 1
  - charObject.compareTo(new Character('b')); //returns 0
  - charObject.compareTo(new Character('c')); //returns -1
  - charObject.compareTo(new Character('d'); //returns -2
  - charObject.equals(new Character('b')); //returns true
  - charObject.equals(new Character('d')); //returns false

The Character Class

java.lang.Character	
+Character(value: char)	Constructs a character object with char value
+charValue(): char	Returns the char value from this object
+compareTo(anotherCharacter: Character): int	Compares this character with another
+equals(anotherCharacter: Character): boolean	Returns true if this character equals to another
+isDigit(ch: char): boolean	Returns true if the specified character is a digit
+isLetter(ch: char): boolean	Returns true if the specified character is a letter
+isLetterOrDigit(ch: char): boolean	Returns true if the character is a letter or a digit
+isLowerCase(ch: char): boolean	Returns true if the character is a lowercase letter
+isUpperCase(ch: char): boolean	Returns true if the character is an uppercase letter
+toLowerCase(ch: char): char	Returns the lowercase of the specified character
+toUpperCase(ch: char): char	Returns the uppercase of the specified character

- This example gives a program that counts the number of occurrence of each letter in a string. Assume the letters are not case-sensitive.

CountEachLetter

The StringBuffer Class (Thread Safe)

- The StringBuffer class is an alternative to the String class. In general, a StringBuffer can be used wherever a string is used (drop-in replacement).
- StringBuffer is mutable, hence more flexible than String.
  - The content of a String object is fixed once the string is created. But you can insert or append new contents into a StringBuffer.

- To construct a StringBuffer:
  - public StringBuffer()
    - No characters, initial capacity 16 characters.
  - public StringBuffer(int length)
    - No characters, initial capacity specified by the length argument.
  - public StringBuffer(String str)
    - Represents the same sequence of charactersas the string argument. The initial capacity of the string buffer is 16 plus the length of the string argument.

Appending Contents to StringBuffer

- To append new contents to buffer:

```
StringBuffer strBuf = new StringBuffer();
strBuf.append("Welcome");
strBuf.append(' ');
strBuf.append("to");
strBuf.append(' ');
strBuf.append("Java");
System.out.println(strBuf);         // strBuf contains "Welcome to Java"
```

java.lang.StringBuffer

+StringBuffer()	Constructs an empty string buffer with capacity 16
+StringBuffer(capacity: int)	Constructs a string buffer with the specified capacity
+StringBuffer(str: String)	Constructs a string buffer with the specified string
+append(data: char[]): StringBuffer	Appends a char array into this string buffer
+append(data: char[], offset: int, len: int): StringBuffer	Appends a subarray in data into this string buffer
+append(v: *aPrimitiveType*): StringBuffer	Appends a primitive type value as string to this buffer
+append(str: String): StringBuffer	Appends a string to this string buffer
+capacity(): int	Returns the capacity of this string buffer
+charAt(index: int): char	Returns the character at the specified index
+delete(startIndex: int, endIndex: int): StringBuffer	Deletes characters from startIndex to endIndex
+deleteCharAt(int index): StringBuffer	Deletes a character at the specified index
+insert(index: int, data: char[], offset: int, len: int): StringBuffer	Inserts a subarray of the data in the array to the buffer at the specified index
+insert(offset: int, data: char[]): StringBuffer	Inserts data to this buffer at the position offset
+insert(offset: int, b: *aPrimitiveType*): StringBuffer	Inserts a value converted to string into this buffer
+insert(offset: int, str: String): StringBuffer	Inserts a string into this buffer at the position offset
+length(): int	Returns the number of characters in this buffer
+replace(int startIndex, int endIndex, String str): StringBuffer	Replaces the characters in this buffer from startIndex to endIndex with the specified string
+reverse(): StringBuffer	Reveres the characters in the buffer
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this buffer
+setLength(newLength: int): void	Sets a new length in this buffer
+substring(startIndex: int): String	Returns a substring starting at startIndex
+substring(startIndex: int, endIndex: int): String	Returns a substring from startIndex to endIndex

Passing Parameters to Main Method

- Method 1 - You can call a regular method by passing actual parameters. Can you pass arguments to main? Sure! For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

- Method 2 (command line) - In the main method below, you can get the arguments from args[0], args[1], ..., args[n], which corresponds to arg0, arg1, ..., argn in the command line.

```
class TestMain {
  public static void main(String[] args) {   ...   }
}
C:\> java TestMain arg0 arg1 arg2 ... argn
```

- Ex: Write a program that will perform binary operations on integers. The program receives three parameters: one operator and two integers.

C:\> java Calculator **2** + **3**

Calculator