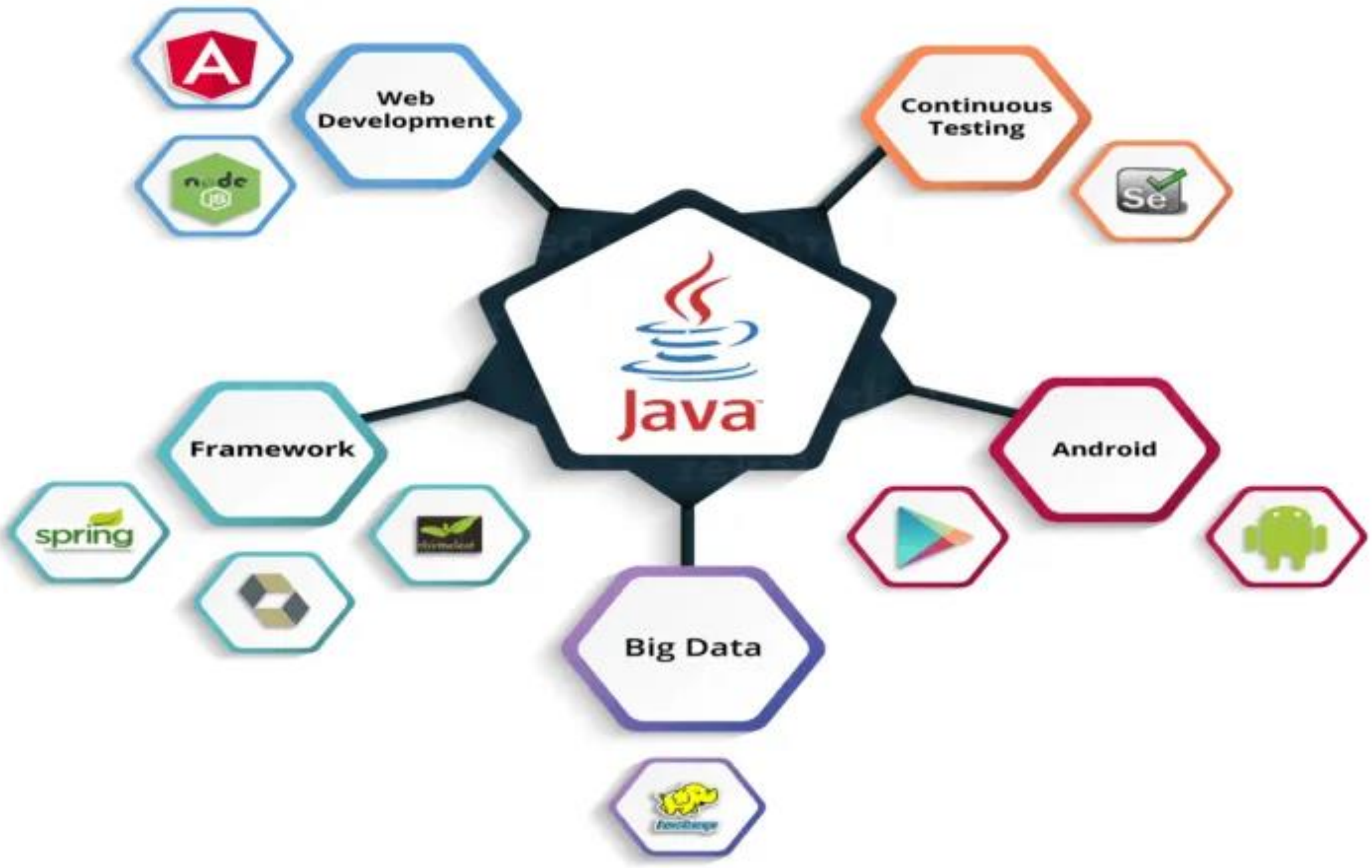
Basic Concepts of Java Programming Part 2

@UD

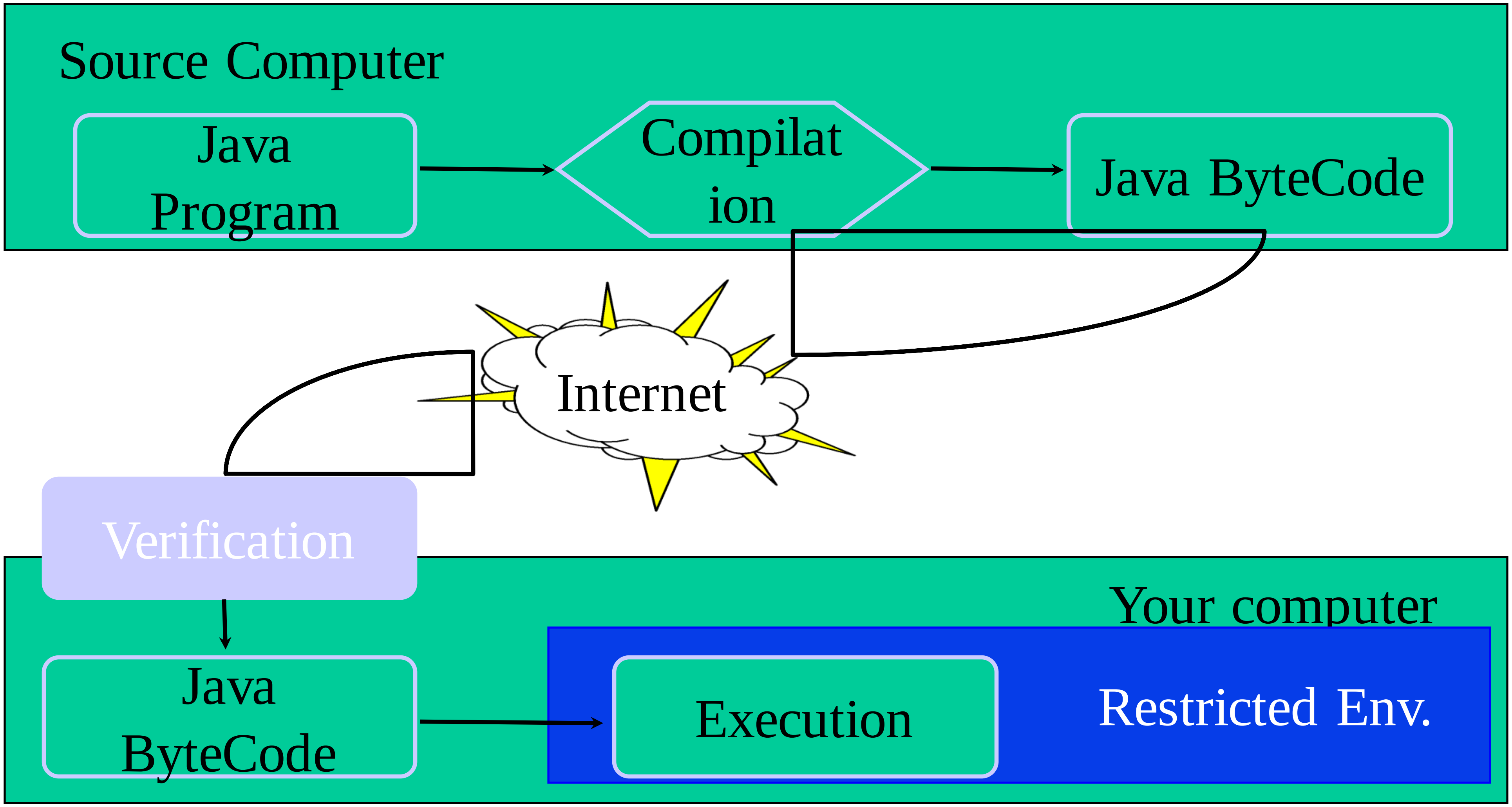
(JVM, Data types, Access specifiers, Operators, Control statements, Array)

Jul 2023

Java Technologies

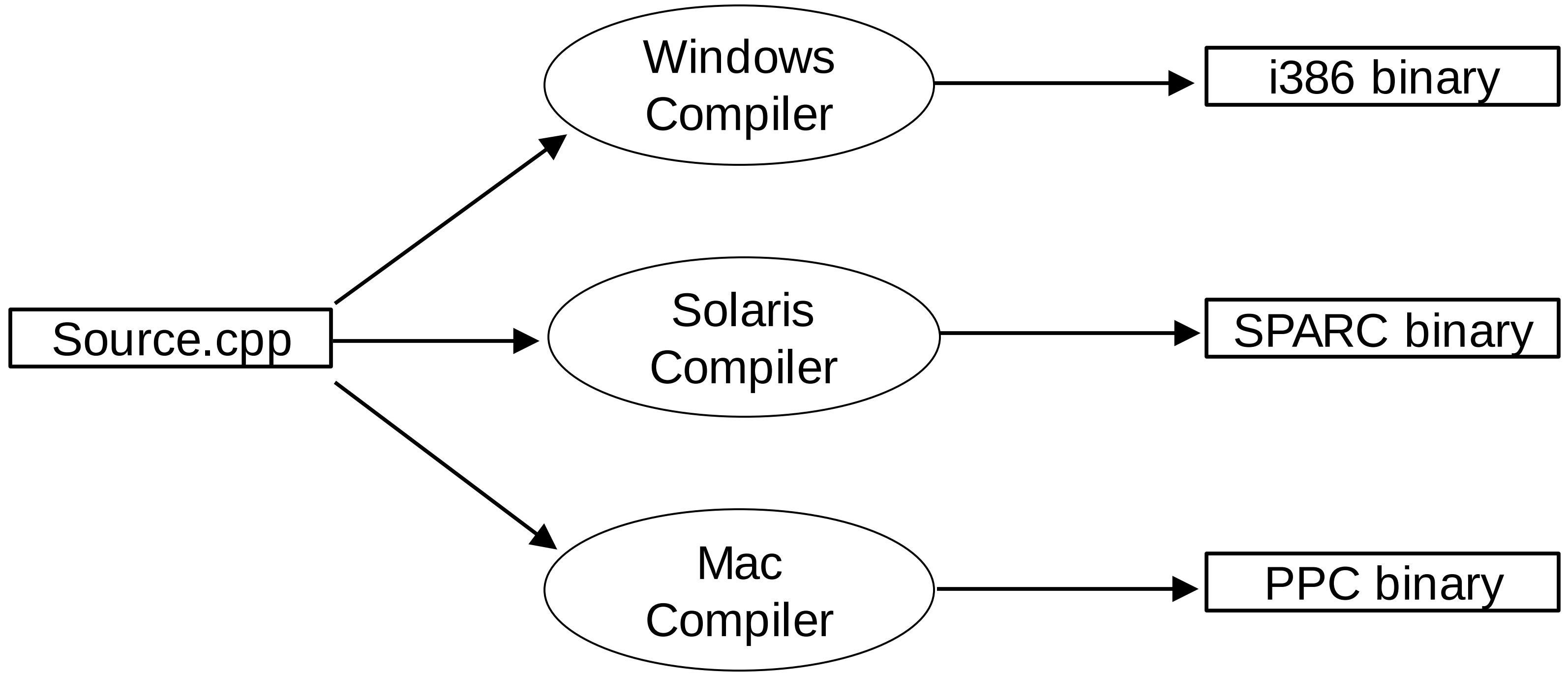


Java Development Basics



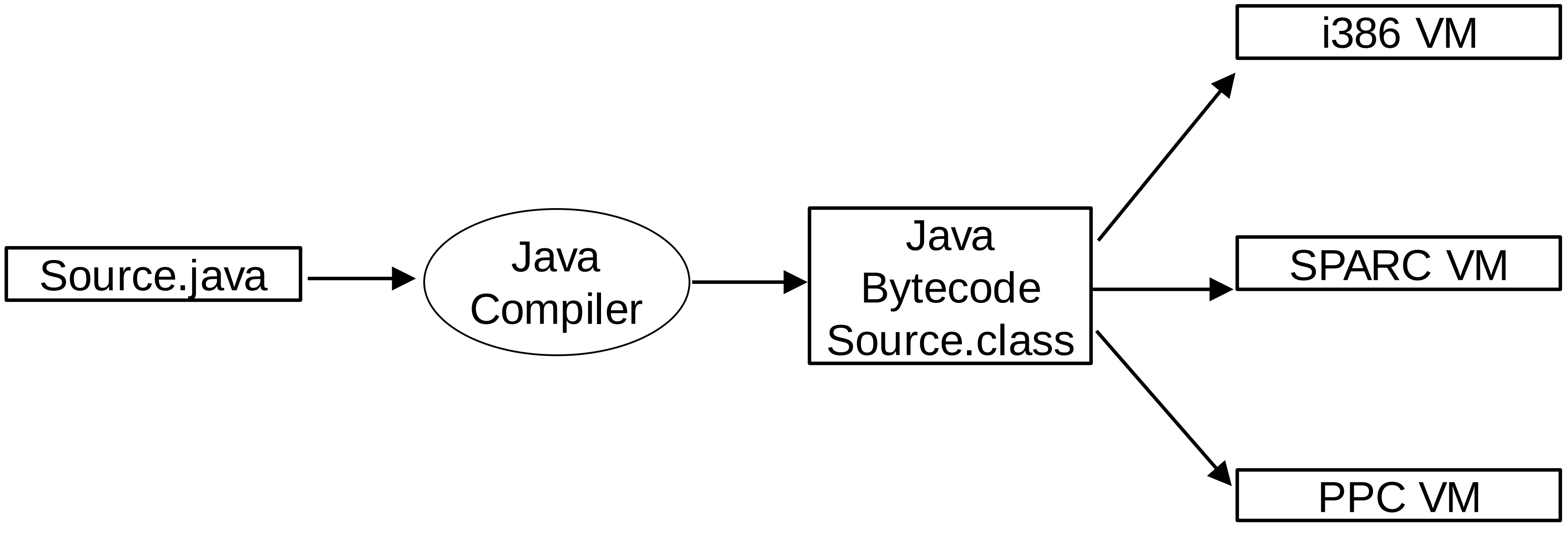
Java Virtual Machine

- Traditionally, source code had to be compiled for the target hardware and OS platform:



Java Virtual Machine

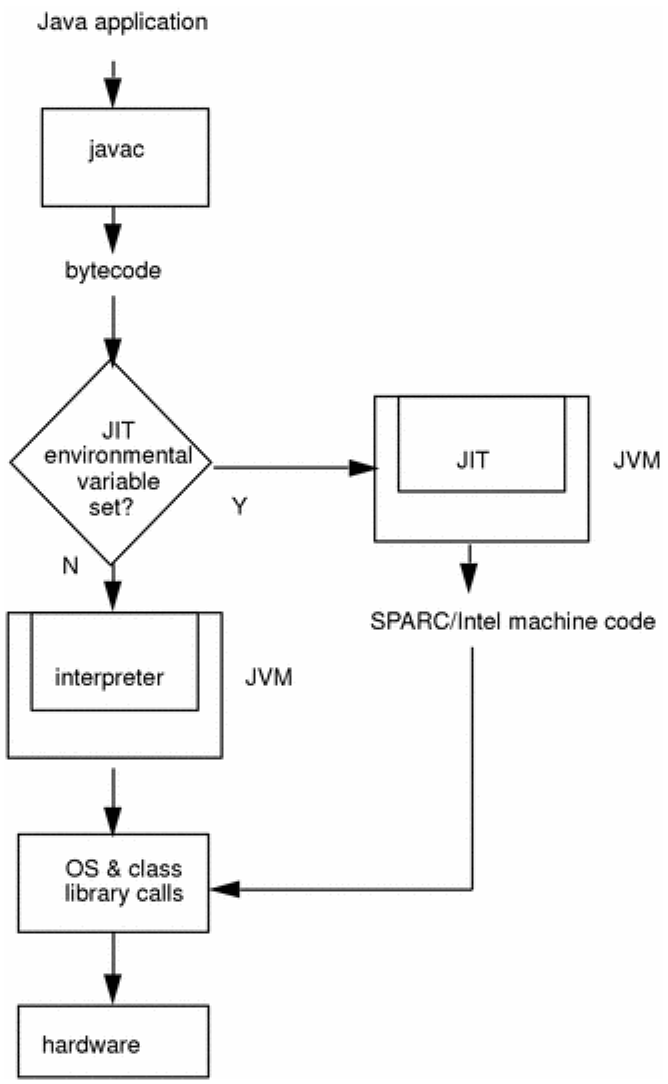
- Java source files (.java) are compiled to Java bytecode (.class)
- Bytecode is interpreted on the target platform within a Java Virtual Machine

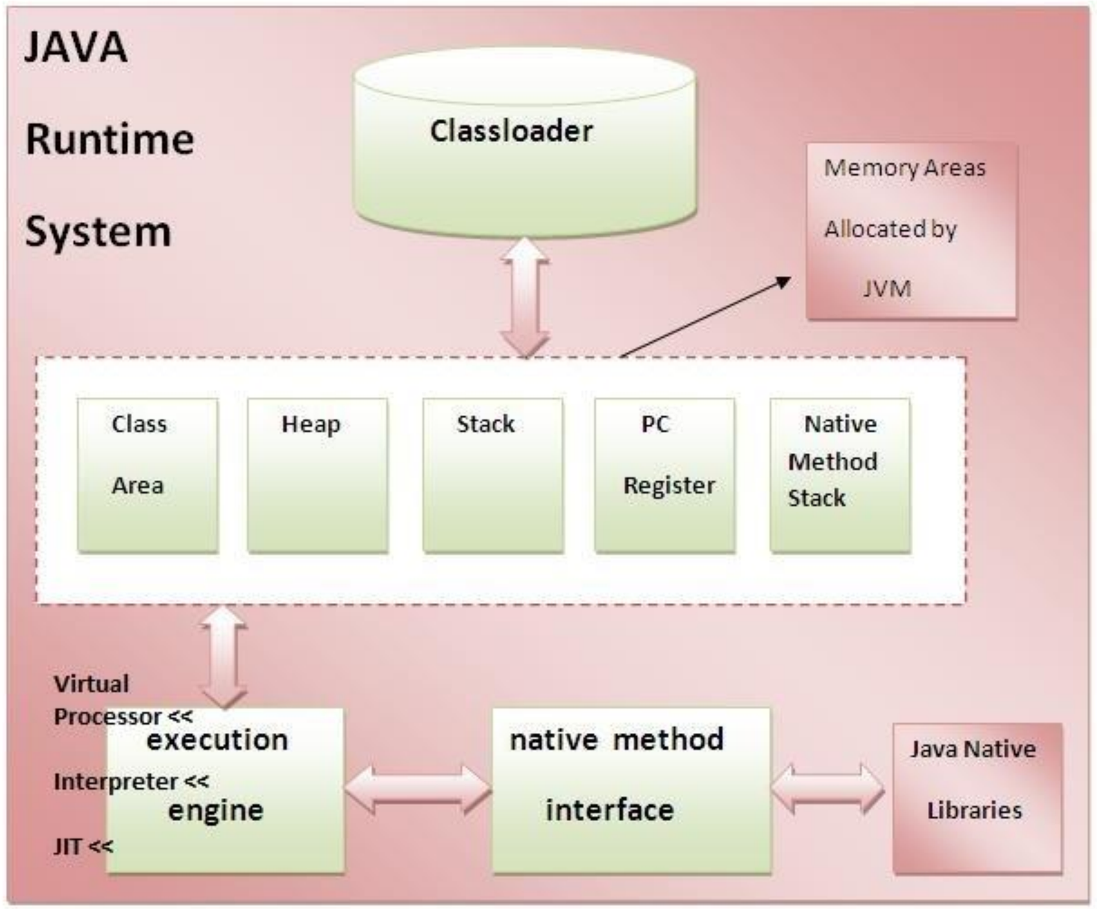


Java Virtual Machine

- Java Virtual Machine (JVM) does more than interpreting bytecode:
  - Class loader loads appropriate java classes (e.g. over the network)
  - Classes verified for legal bytecodes, and not permitted for illegal stack or register usage
  - Security Manager can limit access to resources such as the local file system or the network
  - Any unreferenced memory (Objects) are returned to the system by the Garbage Collector thread
  - Many database servers, application servers, web servers and browsers contain a Java virtual machine
    - eg: Oracle (Database server), Tomcat (web server); WebSphere, BEA Weblogic (app server); and Chrome, Edge, FireFox (Browsers)
  - Languages supported beyond Java: Scala, Groovy, Kotlin; JPython, JRuby, Clojure etc.

JVM - JIT & Memory Management



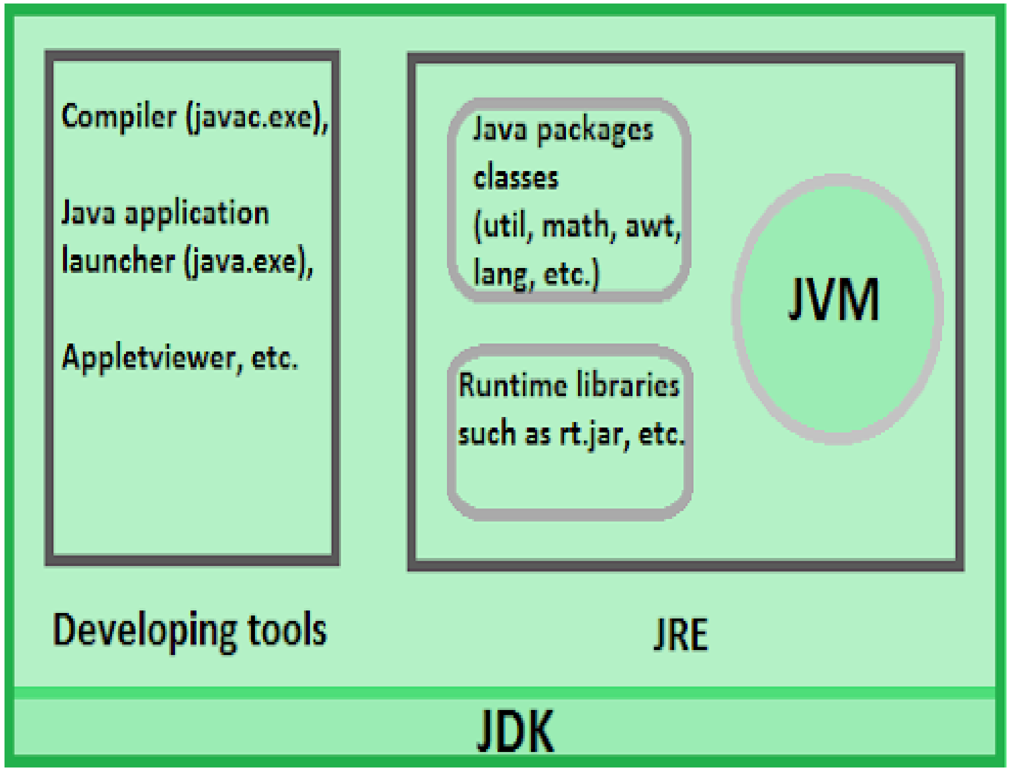


Java Software Development Kit (SDK)

- SDK is Software Development Kit; can be for any language

- Java SDK variations:
  - J2ME - Micro Edition (for handheld and portable devices)
  - J2SE - Standard Edition (PC development)
  - J2EE - Enterprise Edition (Distributed and Enterprise Computing)

- Command line tools
  - javac - Java Compiler
  - java - Java Interpreter
  - appletviewer - Run applets without a browser
  - javadoc - automated documentation generator
  - jdb - Java debugger


Compiler (javac.exe),
Java application launcher (java.exe),
Appletviewer, etc.
Java packages classes (util, math, awt, lang, etc.)
Runtime libraries such as rt.jar, etc.
JVM
Developing tools
JRE
JDK

Integrated Development Environments (IDEs)

- Many IDEs available - some are public domain and some are commercial:
  - Symantic Visual Cafe
  - JBuilder
  - IBM Visual Age
  - Kawa
  - Forte for Java
  - **Netbeans**
  - **Eclipse**
  - **IntelliJ IDEA**

- Many OO modelling tools (such as Together Control Center) include an IDE.

Java Runtime Environment (JRE)

- For mere running of Java programs, JDK is not needed and JRE is sufficient
- Features of JRE:
  - Smaller installer file
  - Less time to download
  - No compiler or development tools.
  - Java VM and support libraries for specified platform.

Obtaining the Java API Documentation

- Java language is quite small.

- Accompanying the language is a Class library (API)
  - Contains core classes.
  - Contains extensions to Java.
  - Mastering Java API takes time

- Extended APIs include: Messaging API, Servlet API, Java2D, Java3D, Java Media Framework, JavaMail and more...

- Refer following link for a comprehensive look at Java SE7 APIs:
https://docs.oracle.com/javase/7/docs/api/

- Look for latest..

Commonly Used Packages

Classes are grouped into logical units called "Packages":

- ✓ Easier to learn and use
- ✓ Developers can also make use of packages to classify their own classes.

Language (general)	java.lang	Common classes used by default for all application development
GUI	java.awt java.awt.event javax.swing	Graphical User Interface, Windowing, Event processing
Misc. Utilities and Collections	java.util	Helper classes, collections
Input/Output	java.io	File and Stream I/O
Networking	java.net	Sockets, Datagrams

Java Keywords

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*	*not used*
**	*added in 1.2*
***	*added in 1.4*
****	*added in 5.0*

Data Types

Type	Meaning	Width (Bits)	Range (Default value)
boolean	True/false	1	false to true (false)
byte	8-bit interger	8	-128 to 127 (0)
char	Character	16 ('\u0000')	'\u0000' to '\uffff'
double	Double-prec. FP	64	1.8*10^308 (0.0d)
float	Single-prec. FP	32	3.4*10^38 (0.0f)
int	Integer	32	-2^31 to 2^31-1 (0)
long	Long integer	64	-2^63 to 2^63-1 (0L)
short	Short integer	16	-32,768 to 32,767 (0)

Notes:

1. Java supports signed numbers, and no need felt for supporting unsign numbers.
2. Double used as default type for decimal values. For precise decimal e.g. currency, use java.math.BigDecimal class
3. For supporting unicode, size of char type is 16 bits.

Access specifiers

- Specifies who / where can access them
- Four types:
  1. **public**: visible to *world*, i.e. all classes within all packages
  2. **private**: visible within class only
  3. **protected**: visible to package and all sub classes
  4. Not specified / **default**: visible to package only

. Match with above

Bed Room

Hall

Reception

Entrance

Access specifiers (cont)

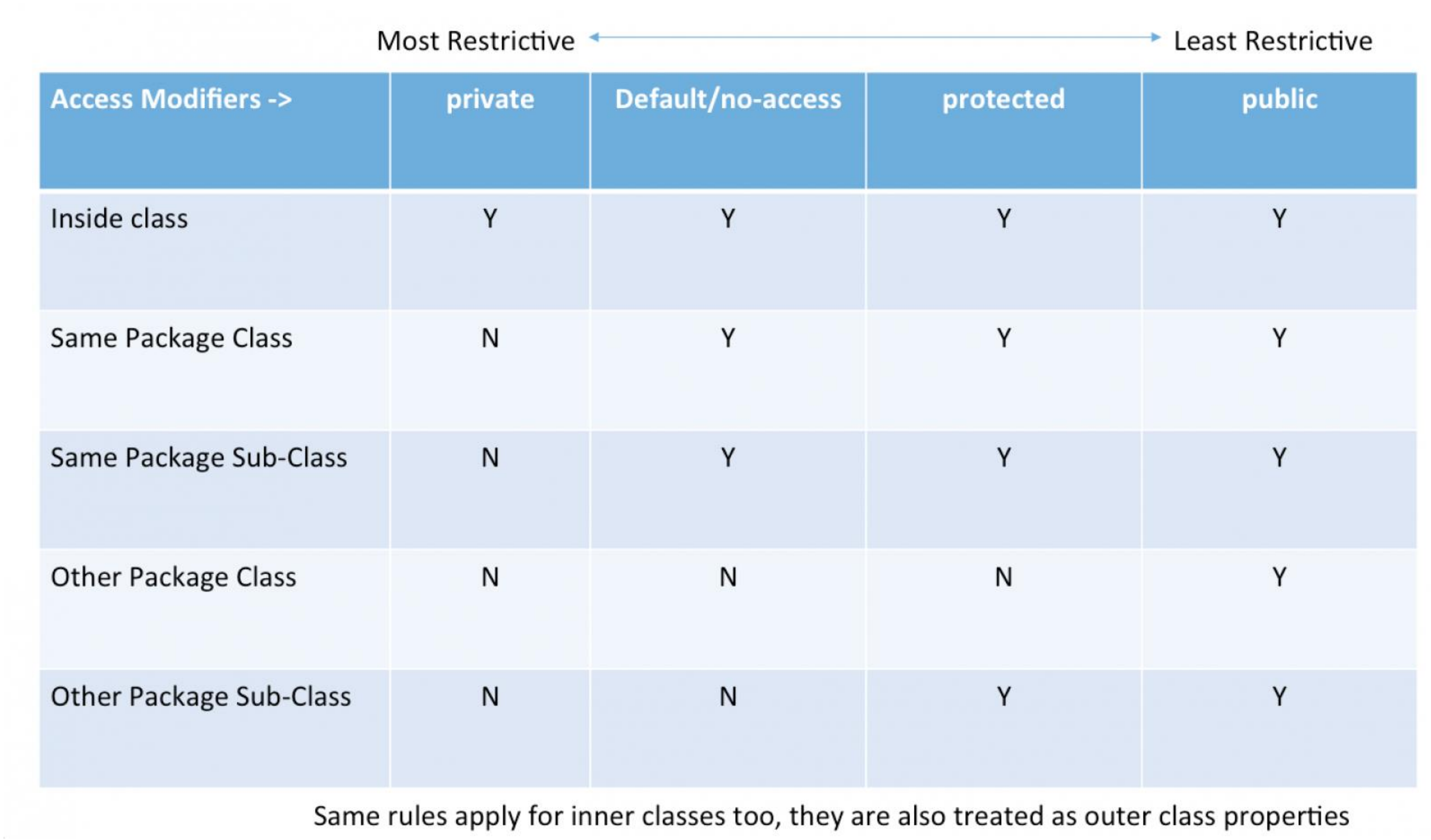
Class level access specifiers (Java classes only)

- **Public**: can be accessed from anywhere
- **Default**: can only be accessed from same package

Member level access specifiers (Java variables & methods)

- **Public / Default**: can be accessed from anywhere
- **Private**: can only be accessed by members
- **Protected**: can be accessed from same package and subclass existing in any package

Class Member Access



Most Restrictive ⟷ Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Same rules apply for inner classes too, they are also treated as outer class properties

Operators with Precedence

Parentheses	(), inside-out
Increment/decrement	++, --, !, unary + or -, ~, (type-cast) from right to left
Multiplicative	*, /, %, from left to right
Additive	+, -, from left to right
Bit-wise operator	>>, << from left to right
Relational	<, >, <=, >=, instanceof from left to right
Equality	==, !=, from left to right
Normal AND / OR, power	& (higher pr.), ^, | (lower pr.) from left to right
Logical / Short-circuit AND	&& from left to right
Logical / Short-circuit OR	||, ?: (lower pr.) from left to right
Assignment	=, +=, -=, *=, /=, %= from right to left

Quiz: `(i) int x=2, y=4; x= x++ + ++y; // print x, y`

`(ii) int x = x << 2; // Effect on x?`

`(iii) Type promotion rules (b/s -> int, l/f/d -> l/f/d)`

Control Statements & Loops

Categories

- Selection *(if, switch)*
- Iteration *(for, while, do-while)*
- Jump *(break, continue, return)*
-

Choice of Loop

- **for loop:** for predefined number of iterations
- **do-while:** for at least one iteration
- **while:** until some condition becomes false

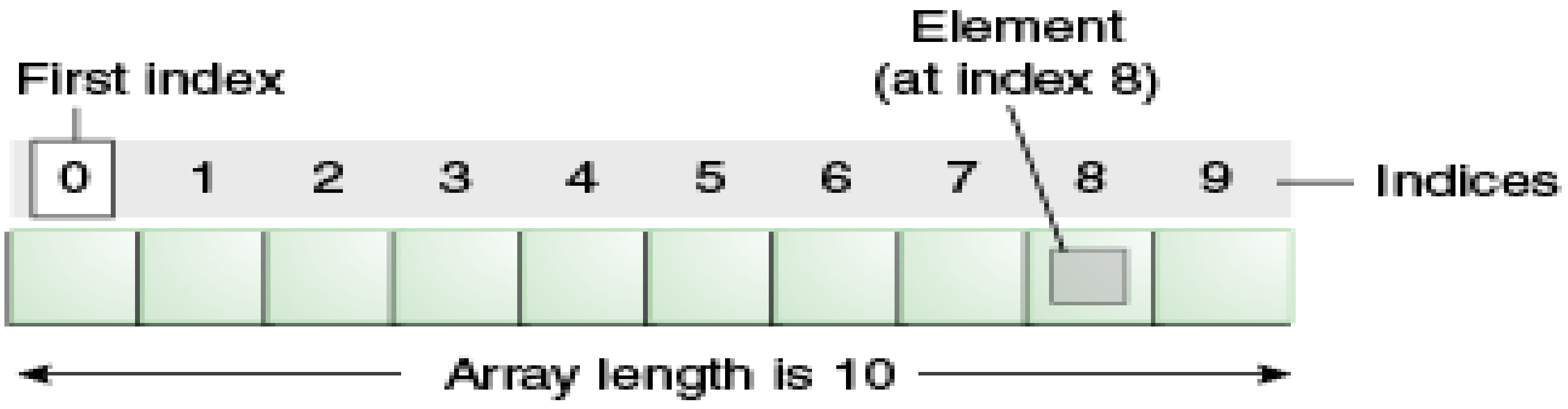
Quiz: (i) Use of NESTED if, if-else-if ladder, switch
(ii) Importance of break in switch statement
(iii) infinite loop using for statement?
(iv) **break label;** ← Difference with goto statement?

Array

Definition:

- Implemented as object where group of contiguous and related data items share a common name
- Particular item is referred using index (subscript)

Types: 1D, 2D, Multi-dimensional



Array

Examples:

```
int nums2[] = { 35, 40, 56, 23, 17, 12, 8, 19, 7,
17 };
int nums1[] = new int[10];
int [] nums1 = new int[10]; // alternative declaration

nums1 = nums2; // Assigning array reference
sum = 0;
for (int i=0; i < nums2.length; i++) sum += nums2[i];
//member length (read-only) in array object.
```

What will happen for 2D array?

```
int nums3[][] = { {35, 40, 56, 23, 17},
                  { 12, 8, 19, 7, 17 } };
```

Enhanced For Loop

Enhanced For or For-each Style (1D array):

```
for (int x: nums2) sum += x; // start to end
```

Enhanced For or For-each Style (2D array):

```
for (int nums3_1D[] : nums3)  // pick up rows
     for ( int x : nums3_1D ) // pick up elements
          sum += x;
```

Limitations:

1) Cannot be used for populating array elements
2) Cannot be used for reading alternate array elements unlike classical-for

break Label

break *label*: *label-ed block must enclose break statement*

Example:

```
public class BreakExample {
  public static void main(String[] args) {
    int[][] intArray= new int[][]{{1,2,3,4,5},{10,20,30,40,50},
{ 6, 8, 30, 12, 91} };
    boolean found = false;
    System.out.println("Searching 30 in specified 2D array..");

    Outer:     // label
    for(int i=0; i < intArray.length ; i++) {  //pick rows
      Inner:
      for(int j=0; j < intArray[i].length; j++){  //pick columns
        if(intArray[i][j] == 30){
          found = true;
          break Outer; }
      }
    }
    if(found == true) ... else …; }
```

Irregular Array

Irregular subarray lengths where lengths are jagged or varies across rows, e.g.

```
public class Animals {
   public static void main(String[] args) {
          String[][] values = new String[2][]; // Create an array of String arrays: a jagged array

          values[0] = new String[2];           // First row with 2-element array (pet animals)
          values[0][0] = "cat"; values[0][1] = "dog";

          values[1] = new String[3];           // Second row with 3-element array (wild animals)
          values[1][0] = "lion"; values[1][1] = "tiger"; values[1][2] = "bear";

          for (String[] array : values) {                // Pick rows
             for (String element : array)                // Pick columns / elements (under the
picked row)
                    System.out.print( element + " " );
             System.out.println();
          }
   }
```

Questions ???

Additional reading: Rules for Access Control & Overriding

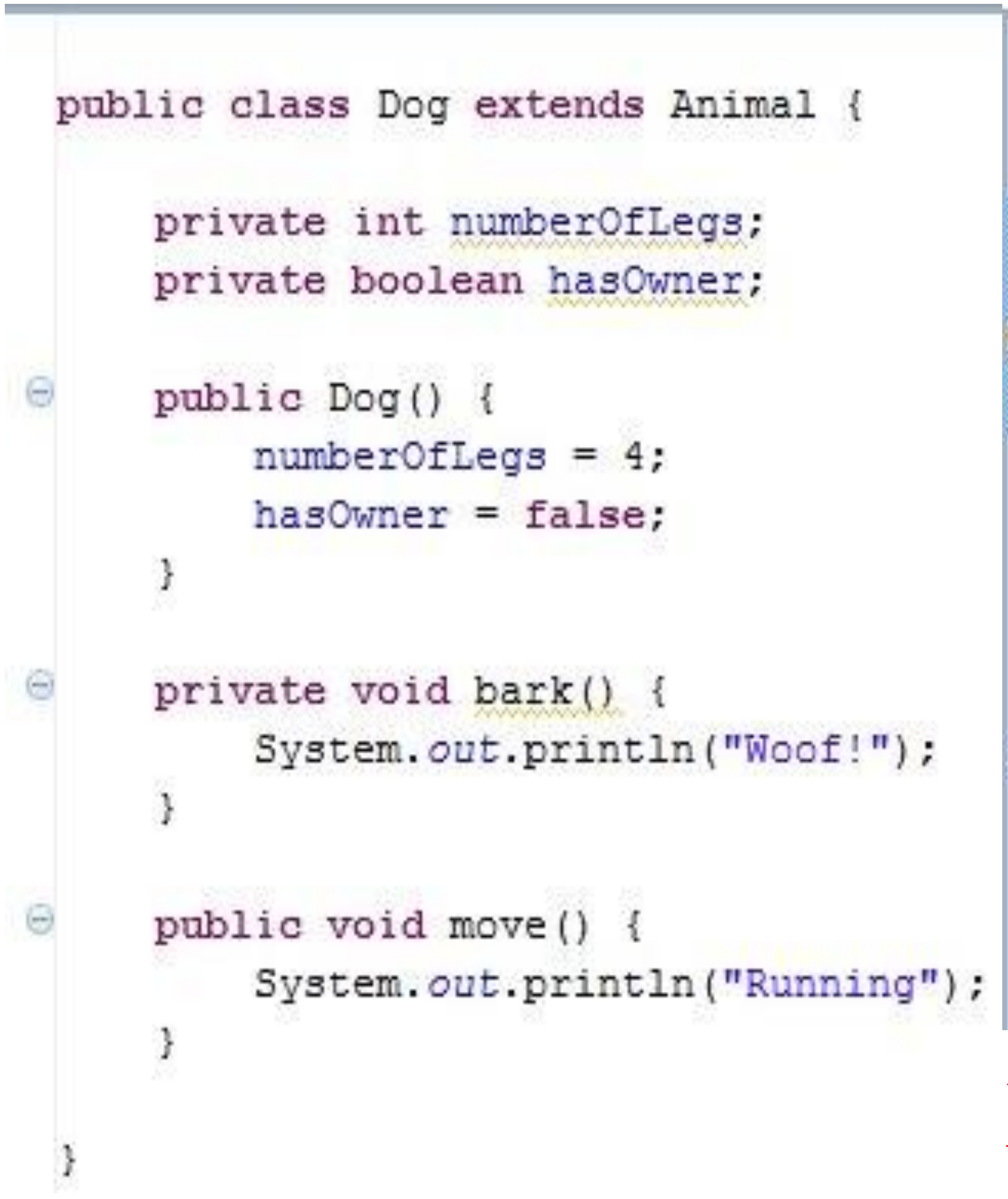
- public method in a Superclass can be overridden only by
    - public method in its Subclass

- Protected methods in a Superclass can be overridden only by
    - protected or public method in its subclass; cannot be private

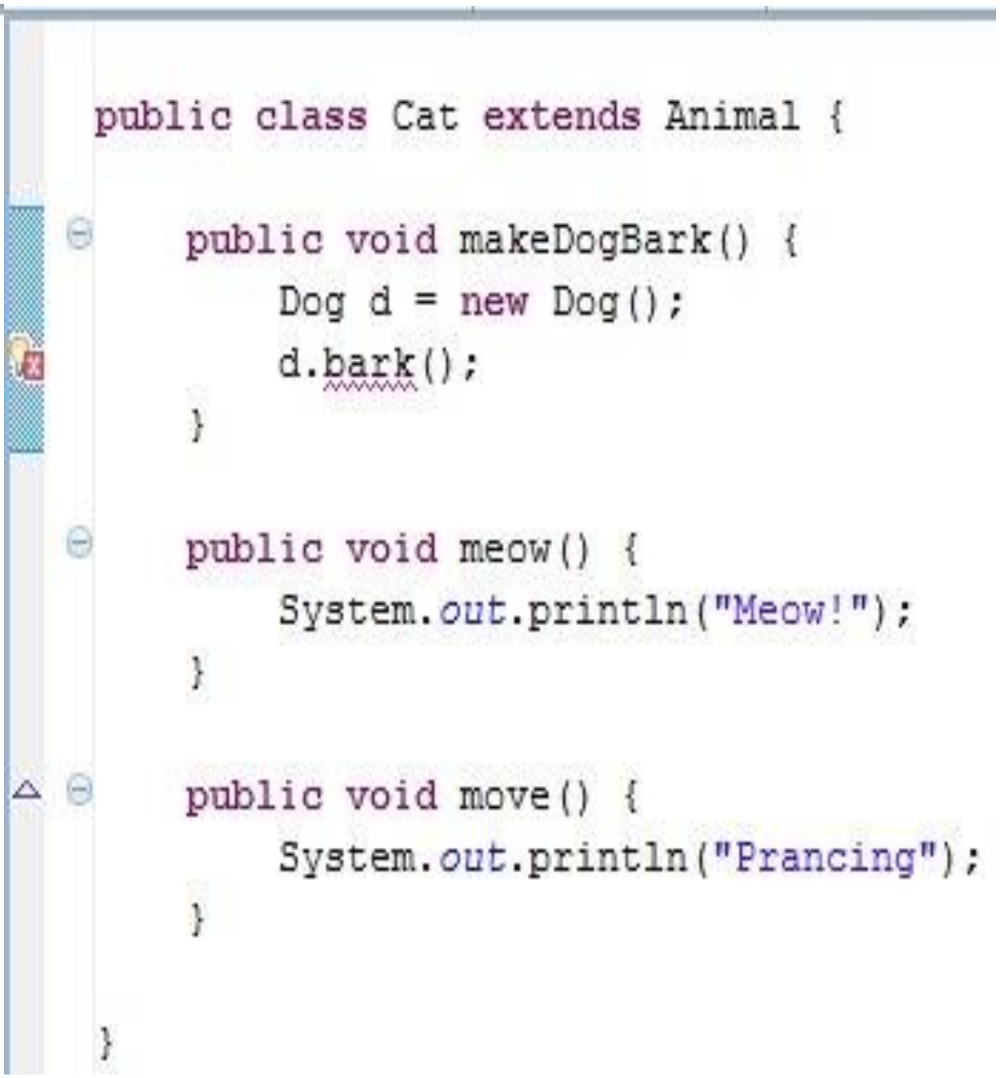
- Default methods in a Superclass can be overridden only by
    - Non-private method in its subclass

- Private method in a Superclass
    - not inherited at all, so there is no rule for them

Additional reading – Puzzles on Access specifiers



```
public class Dog extends Animal {

    private int numberOfLegs;
    private boolean hasOwner;

    public Dog() {
        numberOfLegs = 4;
        hasOwner = false;
    }

    private void bark() {
        System.out.println("Woof!");
    }

    public void move() {
        System.out.println("Running");
    }

}
```

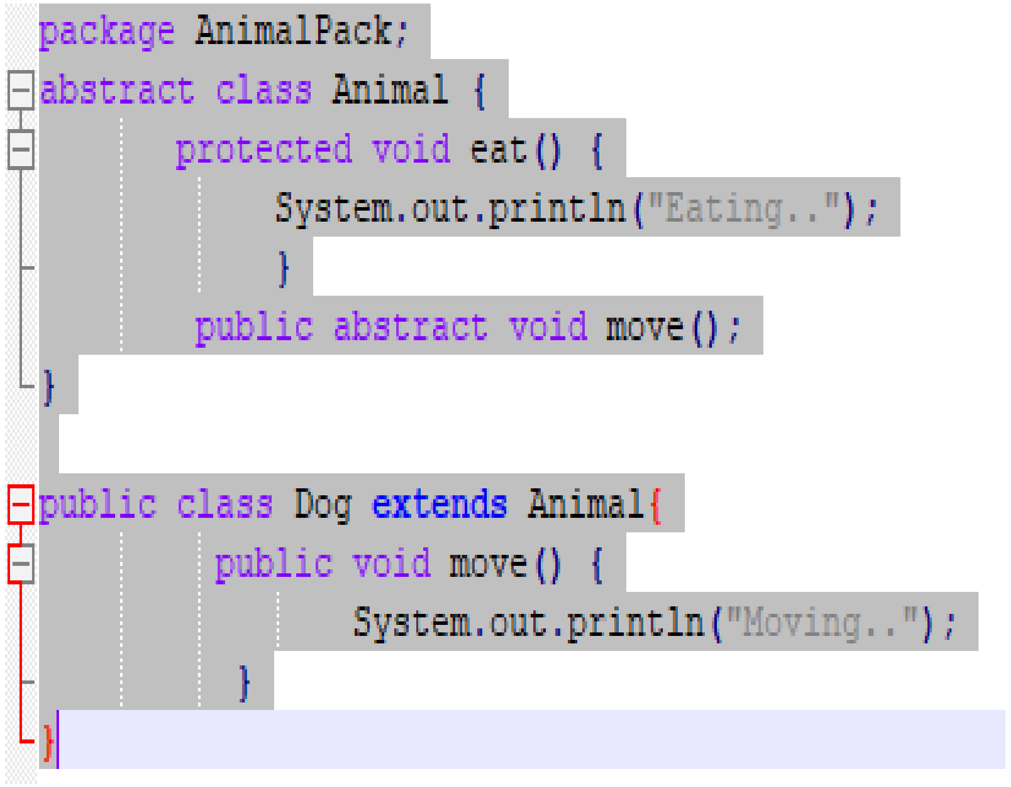


```
public class Cat extends Animal {

    public void makeDogBark() {
        Dog d = new Dog();
        d.bark();
    }

    public void meow() {
        System.out.println("Meow!");
    }

    public void move() {
        System.out.println("Prancing");
    }

}
```

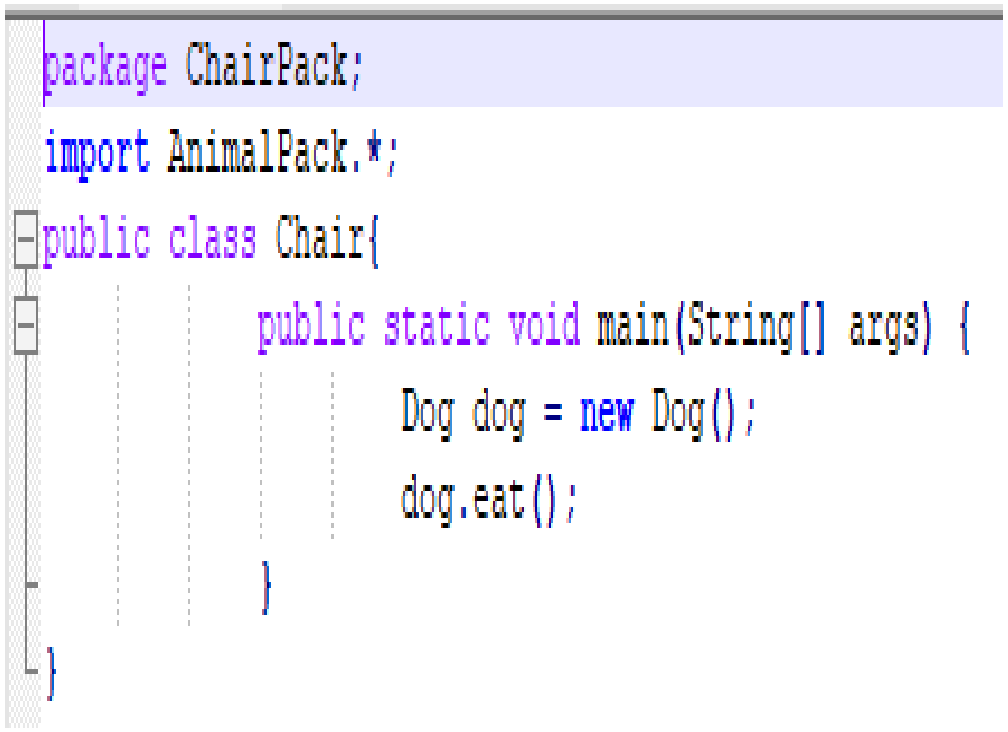
All fine.. now..

Use of bark() in the above example not allowed.. Why? But you can create another dog and make him bark in the same class...

Additional reading: Puzzles on Access specifiers (cont.)



```
package AnimalPack;
abstract class Animal {
        protected void eat() {
            System.out.println("Eating..");
            }
         public abstract void move();
}

public class Dog extends Animal{
           public void move() {
                System.out.println("Moving..");
           }
}
```



```
package ChairPack;
import AnimalPack.*;
public class Chair{
            public static void main(String[] args) {
                    Dog dog = new Dog();
                    dog.eat();
            }
}
```

What's wrong with Chair? Why dog.eat() will be with error?