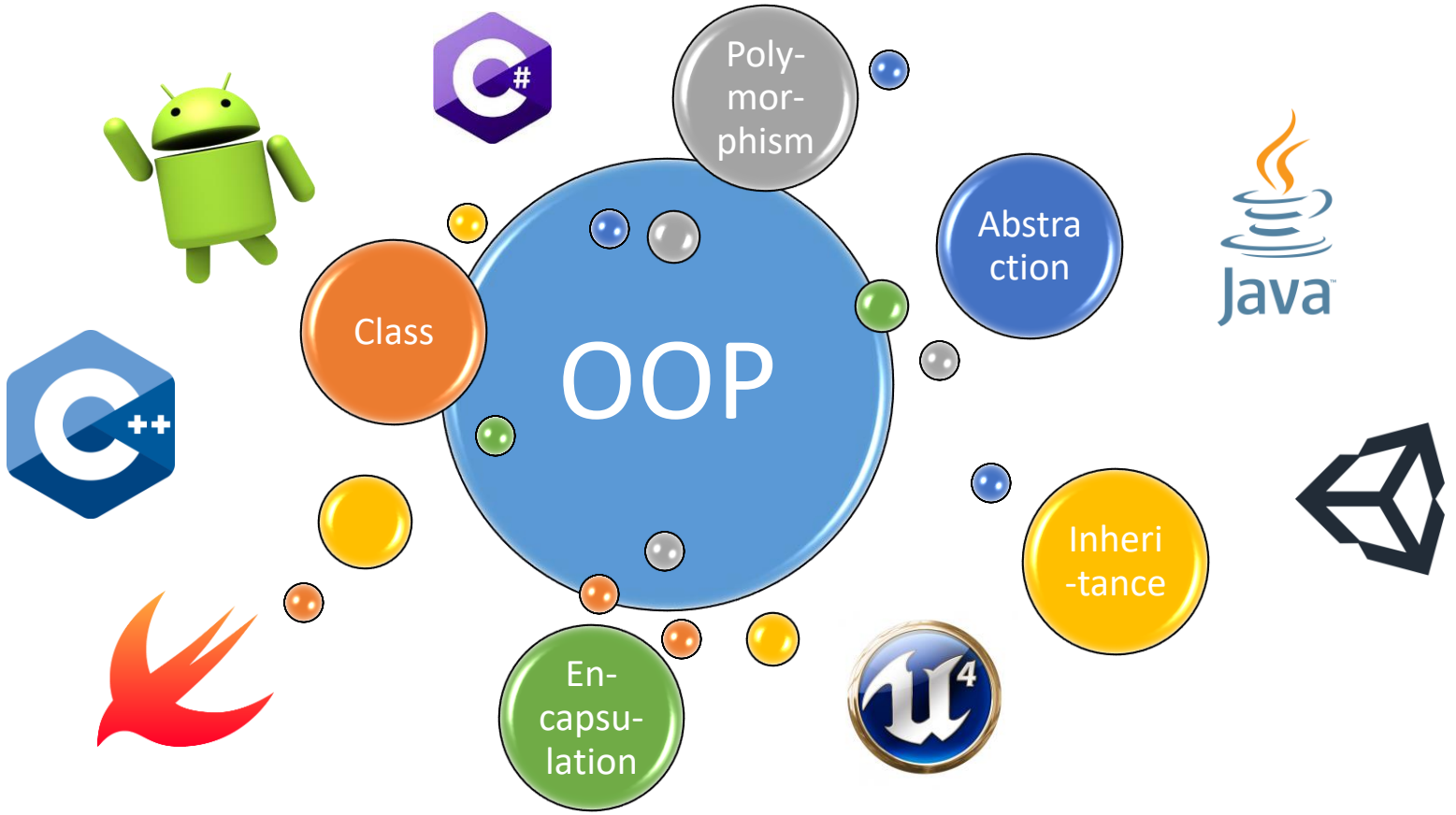
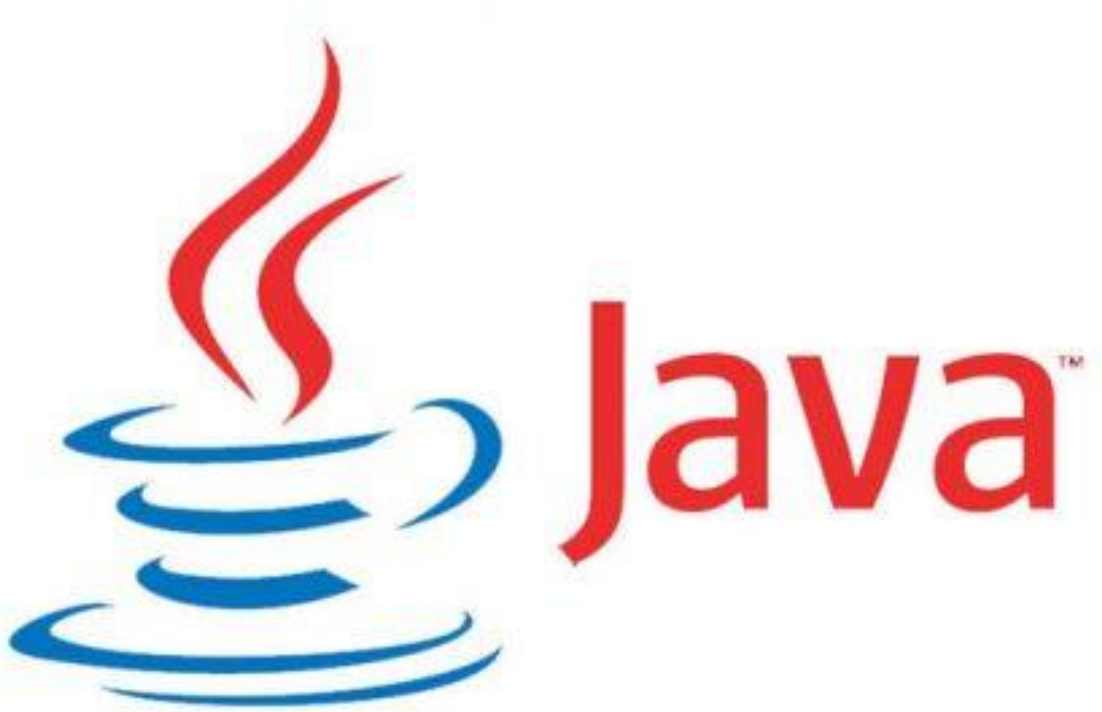


JAVA Basics & OOP

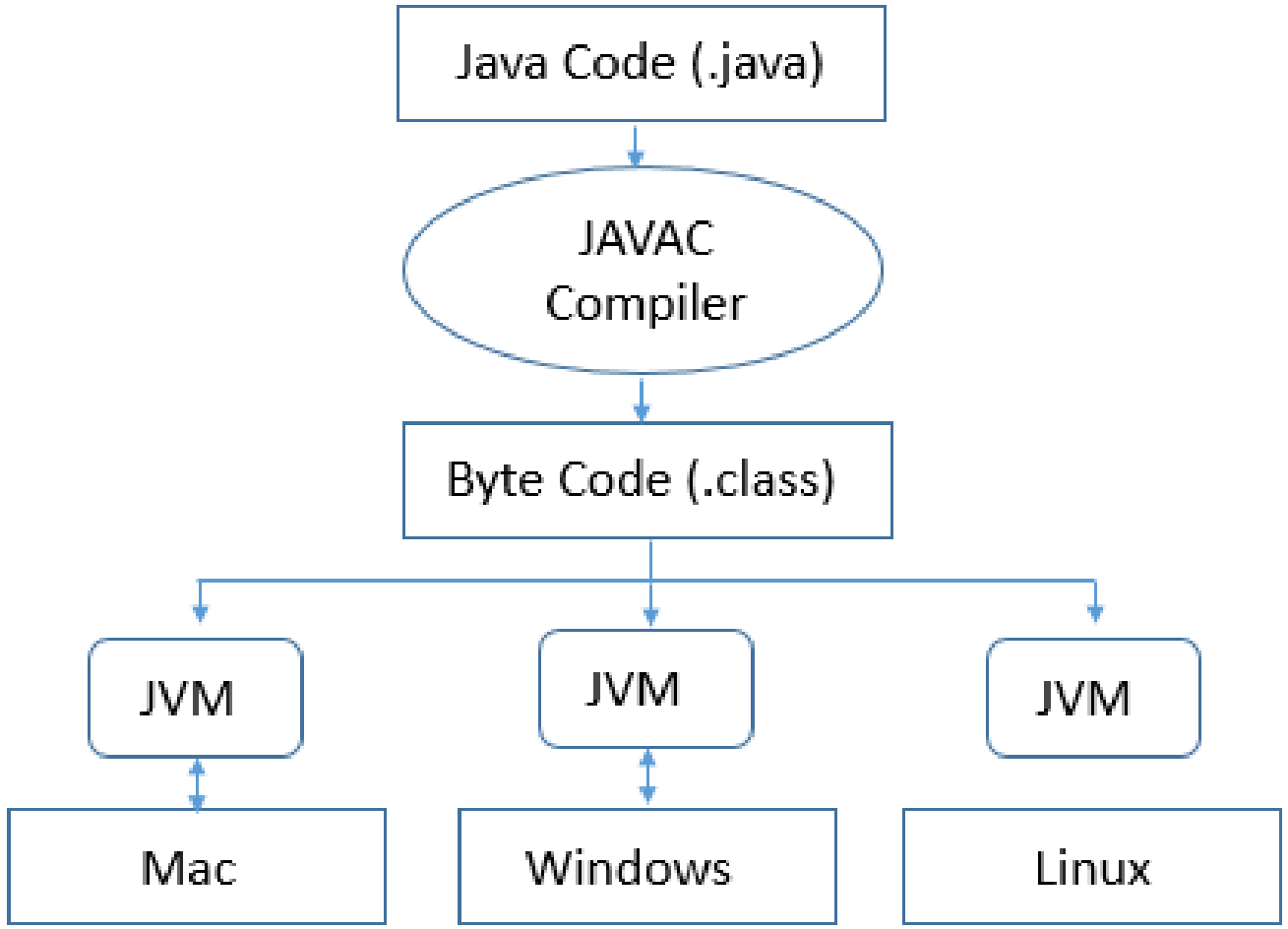
Kuan-Ting Lai

2020/3/14



Write Once, Run Anywhere

Java Virtual Machine (JVM)



http://net-informations.com/java/intro/jvm.htm

Java Overview

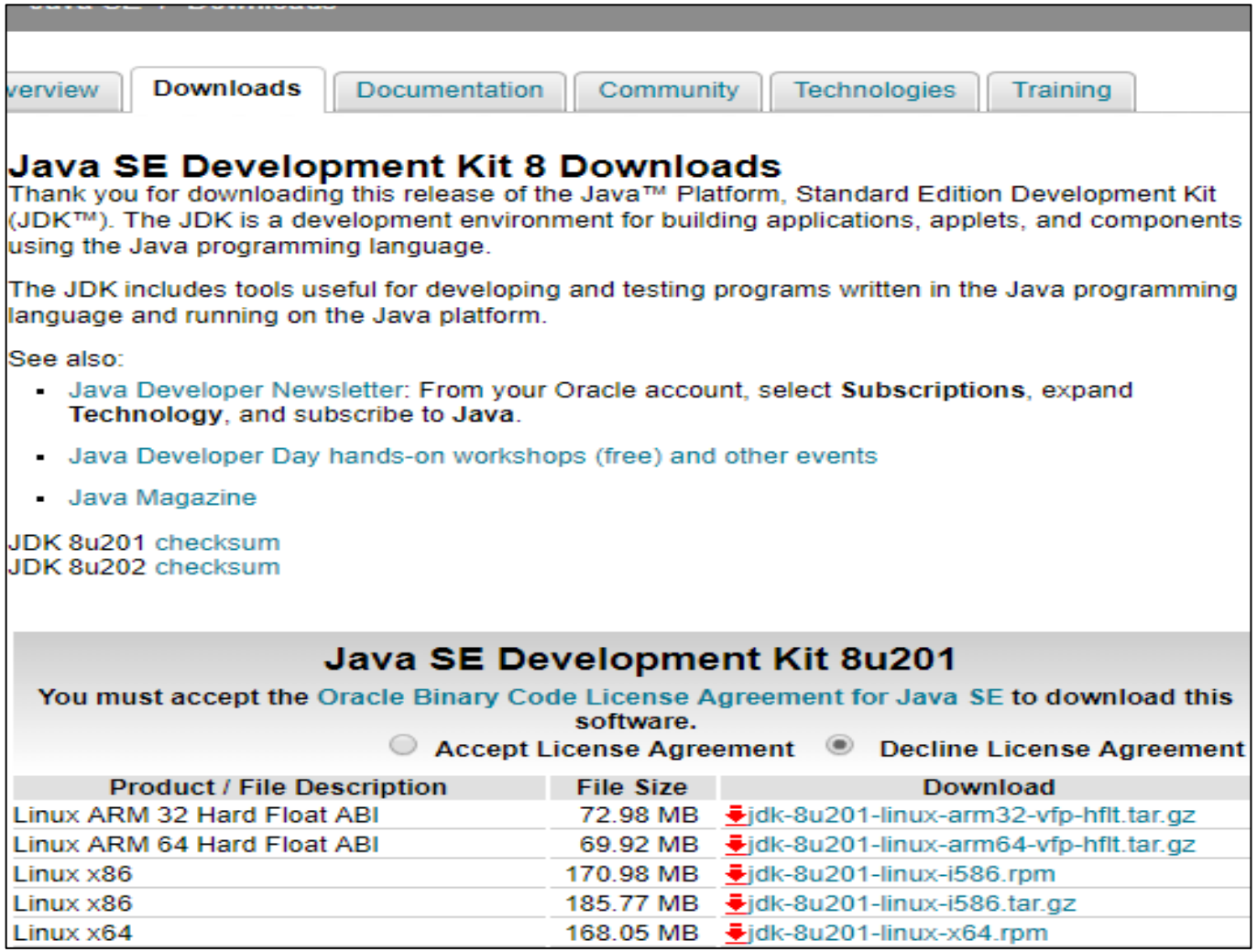
- Simple
- Object-Oriented
- Robust & Secure
- Architecture-neutral and portable
- High Performance
- Interpreted, Threaded, and Dynamic

Brief History of Java

- 1991 – Created by James Gosling. It’s called “Oak” initially
- 1995 – Sun release Java 1.0
- 2007 – Sun made all of Java's core code free and open-source
- 2010 – Sun was bought by Oracle

Install Java Development Kit (JDK)

- Download Java SE Development Kit from Oracle website



verview | **Downloads** | Documentation | Community | Technologies | Training

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u201 checksum
JDK 8u202 checksum

Java SE Development Kit 8u201

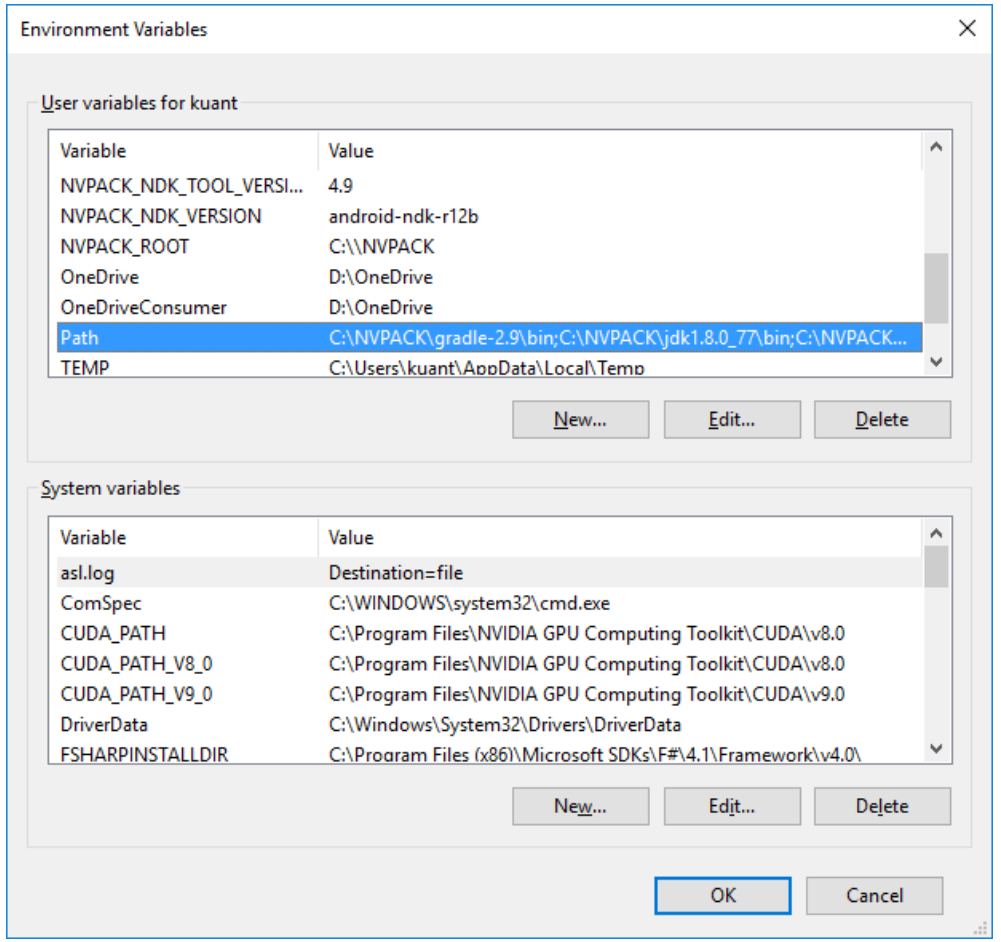
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.98 MB	jdk-8u201-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.92 MB	jdk-8u201-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.98 MB	jdk-8u201-linux-i586.rpm
Linux x86	185.77 MB	jdk-8u201-linux-i586.tar.gz
Linux x64	168.05 MB	jdk-8u201-linux-x64.rpm

Add JDK Folder to “Path” Variable

- Windows
  - 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin’
- Linux
  - export PATH = /path/to/java:$PATH



First Java Program

- Compile and run first java program

  C:\> javac MyFirstJavaProgram.java

  C:\> java MyFirstJavaProgram

  Hello World

```
public class MyFirstJavaProgram {

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

Basic Syntax

- **Class Names** – the first letter should be in Upper Case (CamelCase style)
  - Example: class MyFirstJavaClass
- **Method Names** – start with a Lower Case letter.
  - Example: public void myMethodName()
- **Program File Name** – should exactly match the class name.
  - *MyFirstJavaProgram.java*
- **public static void main(String args[])** – starting point, a mandatory part of every Java program.

Java Class

- Instance variables
  - width, height
- Class variables
  - total_shapes
- Local variables
  - w, h

```
public class Shape {
    int width;
    int height;
    static int total_shapes = 0;

    void setWidth(int w) {
        width = w;
    }

    void setHeight(int h) {
        height = h;
    }
}
```

Constructor

```
public class Shape {
    int width;
    int height;

    public Shape(int a=0, int b=0) {
        width = a; height = b;
    }

    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
}
```

finalize()

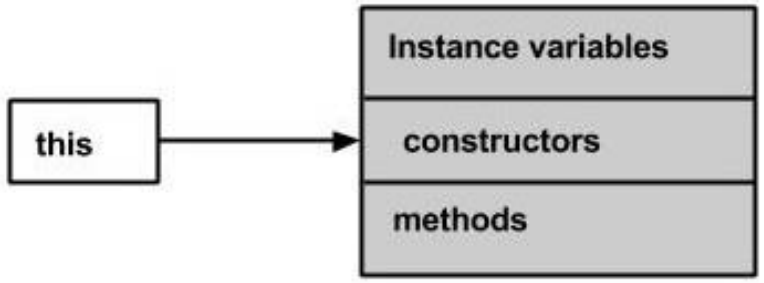
- Java uses a garbage collected language so there is no direct equivalent of a destructor in C++
- There is an inherited method called finalize, but this is called entirely at the discretion of the garbage collector
- For classes that need to explicitly tidy up, the convention is to define a close method and use finalize only for sanity checking (i.e. if close has not been called do it now and log an error).

```
protected void finalize( ) {
    // finalization code here
}
```

https://stackoverflow.com/questions/171952/is-there-a-destructor-for-java

“this” pointer

- **this** is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor




```
class Student {
    int age;

    Student() {
        this(20);
    }

    Student(int age) {
        this.age = age;
    }
}
```

Passing Parameters by Value

- Parameters cannot be passed by reference!
- Objects are always accessed by reference

```
public class SwappingExample {
    public static void main(String[] args) {
        int a = 30; int b = 45;
        System.out.println("Before swapping, a = " + a + " and b = " + b);
        // Invoke the swap method
        swapFunction(a, b);
        System.out.println("\n**Before and After swapping values will be same here**:");
        System.out.println("After swapping, a = " + a + " and b = " + b);
    }
    public void swapFunction(int a, int b) {
        System.out.println("Before swapping(Inside), a = " + a + " b = " + b);
        // Swap n1 with n2
        int c = a;
        a = b;
        b = c;
        System.out.println("After swapping(Inside), a = " + a + " b = " + b);
    }
}
```

Compile & Run Your Second Java Program

- C:\> javac Shape.java
- C:\> javac ShapeTest.java
- C:\> java ShapeTest

  7.700000000000001

Shape.java

```
public class Shape {
    double width = 0;
    double height = 0;

    Shape(double a, double b) {
        width = a; height = b;
    }

    double area() {
        return width * height;
    }
};
```

ShapeTest.java

```
import java.io.*;

public class ShapeTest
{
    public static void main(String args[])
    {
        Shape shape = new Shape(3.5, 2.2);
        System.out.println(shape.area());
    }
}
```

Java Primitive Data Types

- byte (8-bit)
- short (16-bit)
- int (32-bit)
- long (64-bit)
- float (32-bit)
- double (64-bit)
- boolean (1-bit)
- char (16-bit Unicode character)

Java Modifiers

- Access Control Modifiers
  - public, private, protected
- Non-Access Modifiers
  - static, final, abstract
  - synchronized, volatile

```
public class className {
    // ...
}
private boolean myFlag;
static final double weeks = 9.5;
protected static final int BOXWIDTH = 42;

public static void main(String[] arguments) {
    // body of method
}
```

Java Operators

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Loop Control Statements

- while
- for
- do while
- for(declaration : expression)

```
public class Test {
      public static void main(String args[]) {
            int [] numbers = {10, 20, 30, 40, 50};

            for(int x : numbers ) {
               System.out.print( x );
               System.out.print(",");
            }
            System.out.print("\n");
            String [] names = {"James", "Larry", "Tom", "Lacy"};

            for( String name : names ) {
               System.out.print( name );
               System.out.print(",");
            }
      }
}
```

Decision Making

- if statement

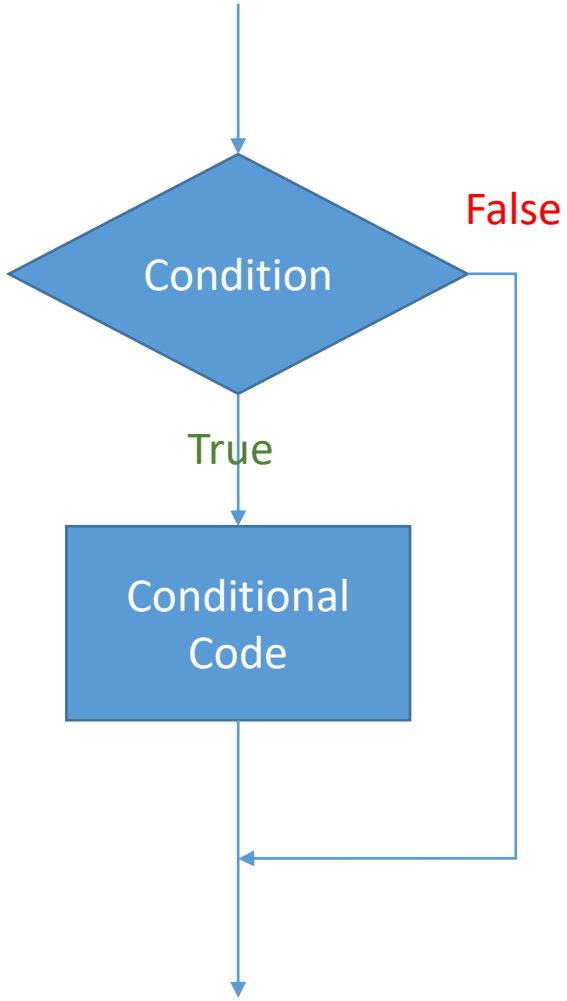
```
if (….) {
    …
} else if (….) {
    …
} else {
    …
}
```

- switch statement

```
switch (….) {
    case 0: …
        break;
    case 0: …
        break;
    default: …
}
```

- ? : operator

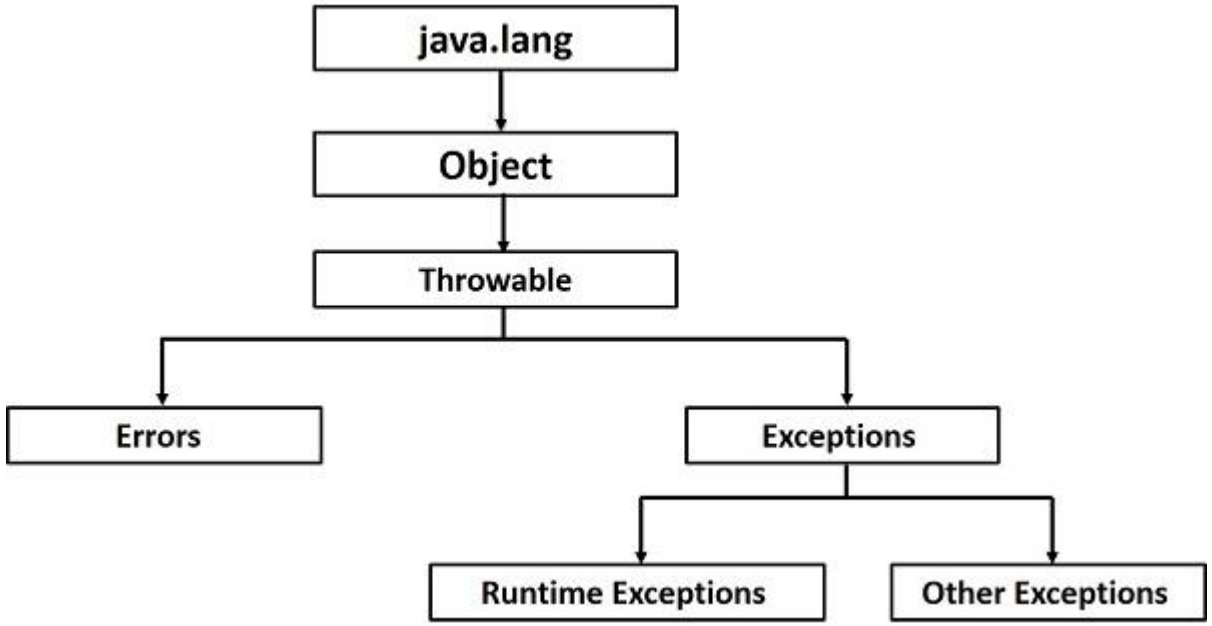
```
Exp1 ? Exp2 : Exp3;
```



Handling Exceptions

- Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs
  - Example: JVM is out of memory
- Two main subclasses: IOException class and RuntimeException Class

```
try {
    // Protected code
}
catch (ExceptionName e1) {
    // Catch block
}
```



Catching Multiple Exceptions

```
try {
    file = new FileInputStream(fileName);
    x = (byte)file.read();
}
catch (FileNotFoundException f) {
    f.printStackTrace();
    return -1;
}
catch (IOException i) {
    i.printStackTrace();
    return -1;
}
finally {
    System.out.println("The finally statement is executed");
}
```

Java String

- Initialize a String

```
public class StringDemo {
    public static void main(String args[]) {
        char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
        String helloString = new String(helloArray);
        System.out.println( helloString );
    }
}
```

- Concatenate String

```
public class StringDemo {
    public static void main(String args[]) {
        String string1 = "saw I was ";
        System.out.println("Dot " + string1 + "Tod");
    }
}
```

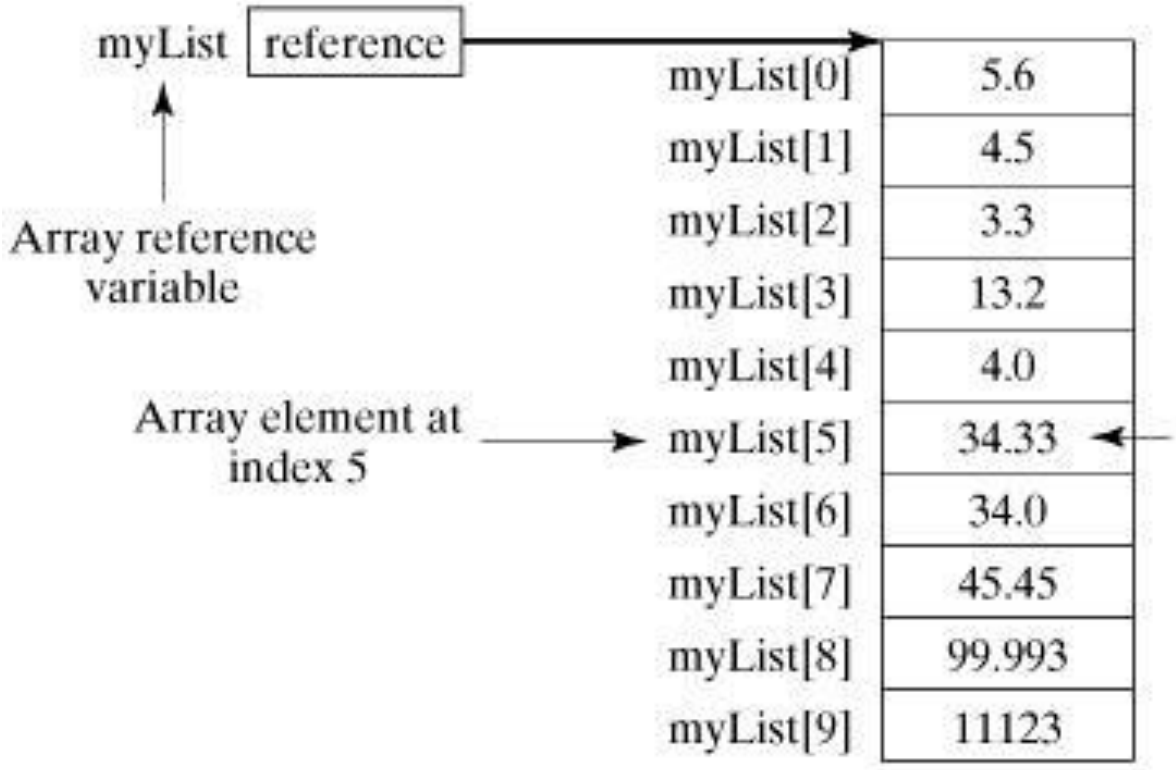
String Methods

Method Name	Description
int length()	Returns the length of this string.
char charAt(int index)	Returns the character at the specified index.
int compareTo(String anotherString)	Compares two strings lexicographically.
byte[] getBytes(String charsetName)	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
int indexOf(int ch)	Returns the index within this string of the first occurrence of the specified character.
int lastIndexOf(int ch)	Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
String replace(char oldChar, char newChar)	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String substring(int beginIndex)	Returns a new string that is a substring of this string.
String[] split(String regex)	Splits this string around matches of the given regular expression.
String trim()	Remove leading and trailing whitespace.

Java Array

- double[] myList = new double[10];

```
public class TestArray {
        public static void main(String[] args) {
            double[] myList = {1.9, 2.9, 3.4, 3.5};

              // Print all the array elements
            for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }
}
```



Java Date

- Date and Simple Date Format

```
import java.util.Date;

public class DateDemo {
        public static void main(String args[]) {
                // Instantiate a Date object
                Date date = new Date();

                // display time and date using toString()
                System.out.println(date.toString());
        }
}
```

```
C:> on May 04 09:51:52 CDT 2009
```

```
import java.util.*;
import java.text.*;

public class DateDemo {
        public static void main(String args[]) {
                Date dNow = new Date( );
                SimpleDateFormat ft =
                new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
                System.out.println("Current Date: " + ft.format(dNow));
        }
}
```

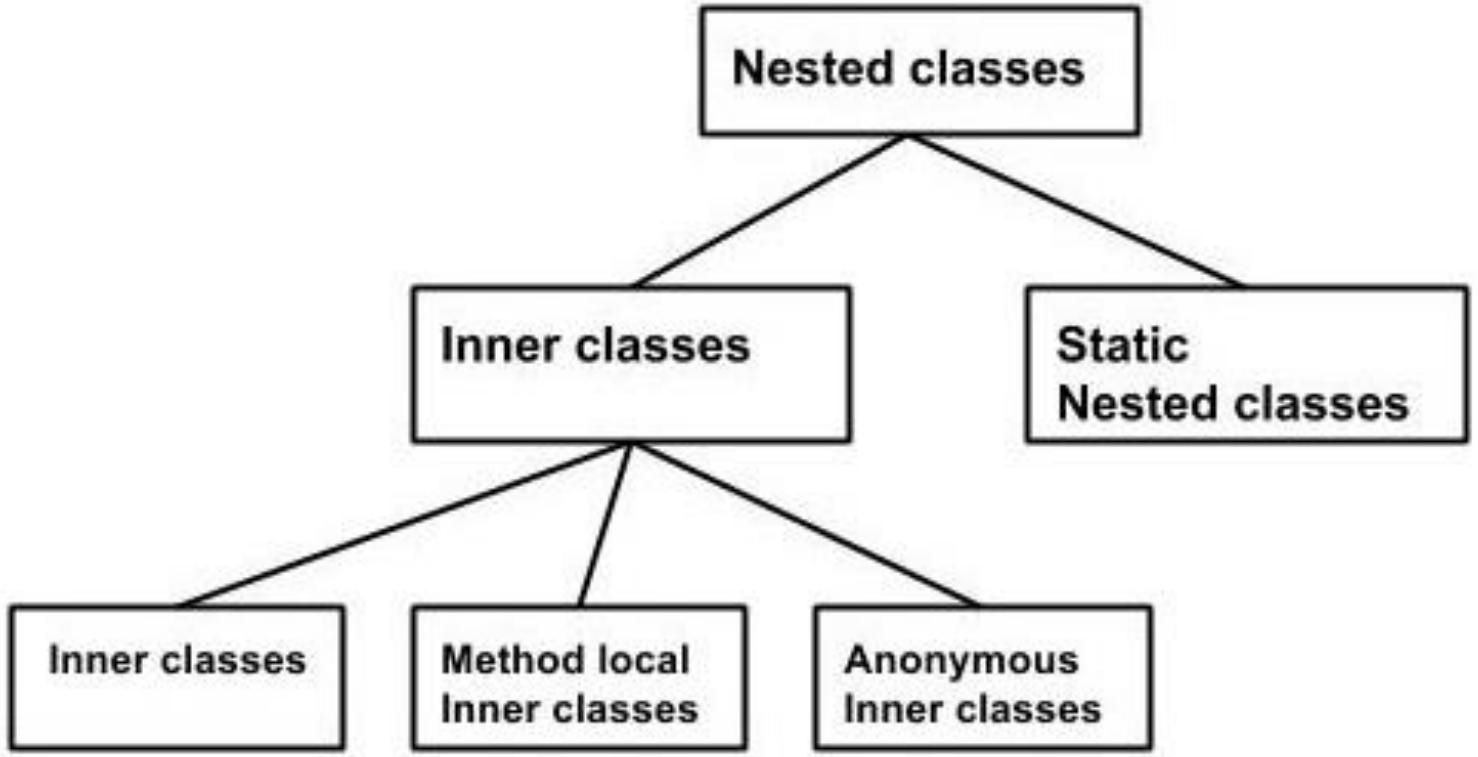
```
C:> Current Date: Sun 2004.07.18 at 04:14:09 PM PDT
```

Java Files and I/O

```
import java.io.*;
public class CopyFile {
    public static void main(String args[]) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
            int c;
            while ((c = in.read()) != -1)
                out.write(c);
        }
        finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```

Inner Class

- Writing a class within another



Java Enum

- A special class

```
enum Level { LOW, MEDIUM, HIGH }

public class MyClass {
      public static void main(String[] args) {
            Level myVar = Level.MEDIUM;
            System.out.println(myVar);
      }
}
```

OOP in Java

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Interfaces
- Packages

Inheritance

- **extends** is the keyword used to inherit the properties of a class

```
class SuperClass {
.....
.....
}
class SubClass extends SuperClass {
.....
.....
}
```

Keyword “super”

- Access parent (super) class

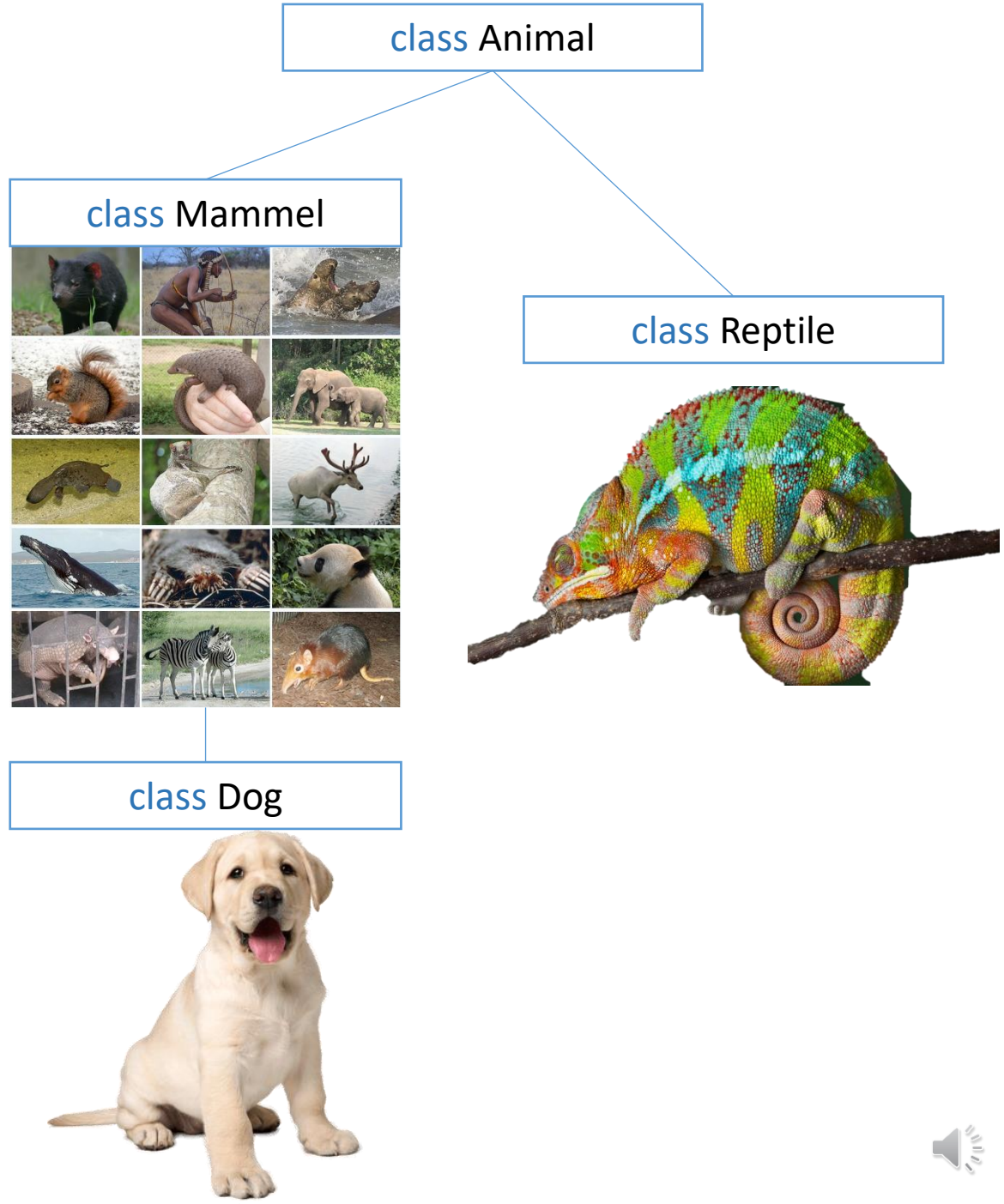
```
class Superclass {
    int age;
    Superclass(int age) {
        this.age = age;
    }
    public void getAge() {
        System.out.println("The value of the variable named age in super class is: " + age);
    }
}
public class Subclass extends Superclass {
    Subclass(int age) {
        super(age);
    }
    public static void main(String args[]) {
         Subclass s = new Subclass(24);
        s.getAge();
    }
}
```

https://stackoverflow.com/questions/180601/using-super-in-c

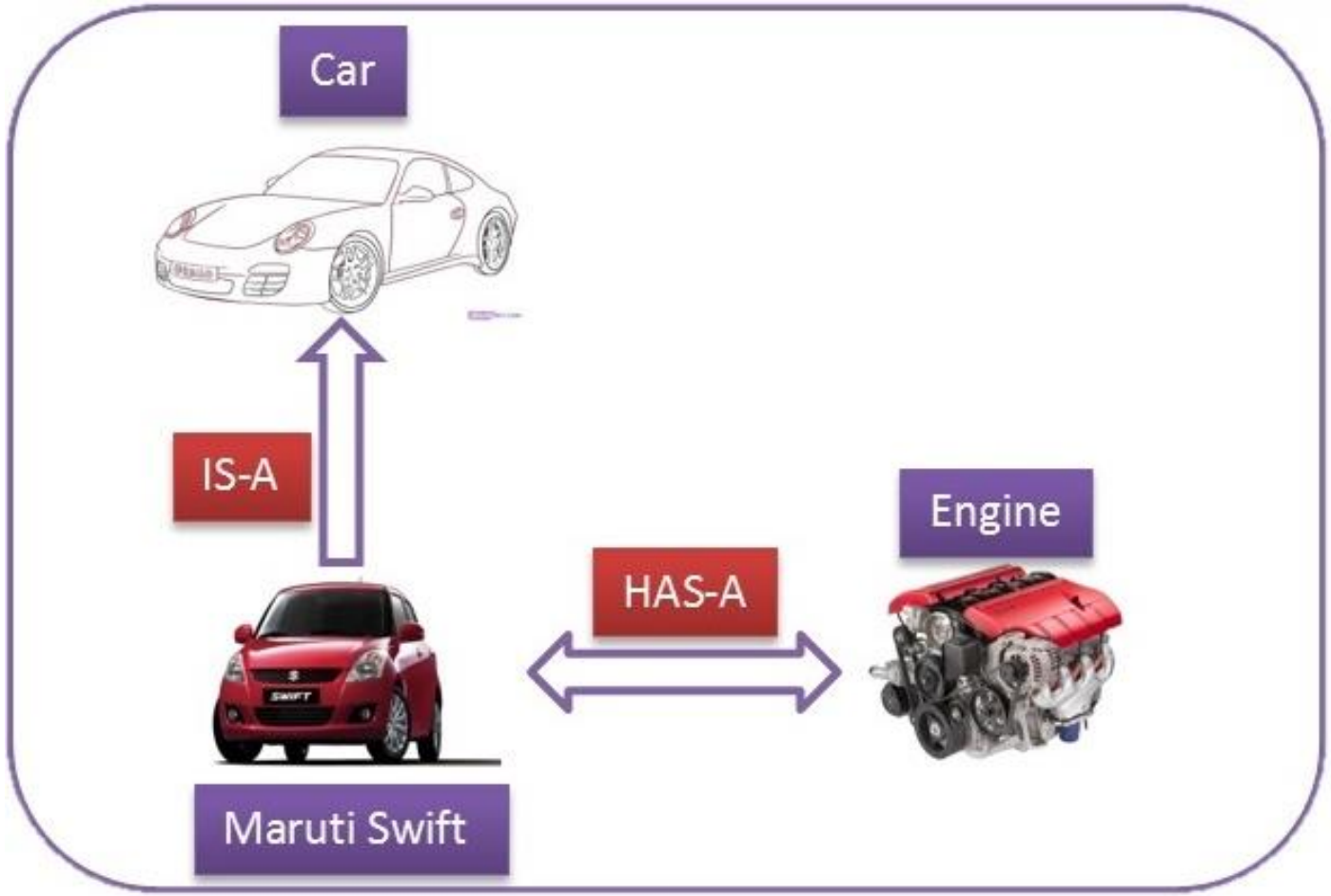
IS-A Relationship

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

```
public class Animal {
}

public class Mammal extends Animal {
}

public class Reptile extends Animal {
}

public class Dog extends Mammal {
}
```



HAS-A Relationship

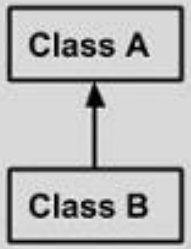


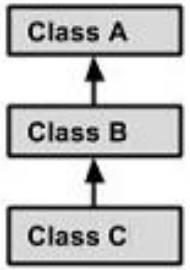
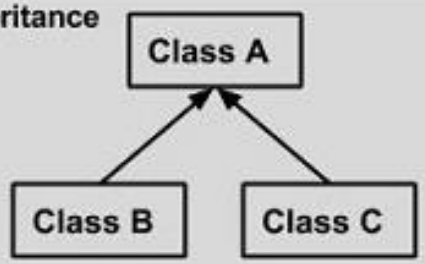
https://ramj2ee.blogspot.com/2015/08/java-tutorial-inheritance-has.html

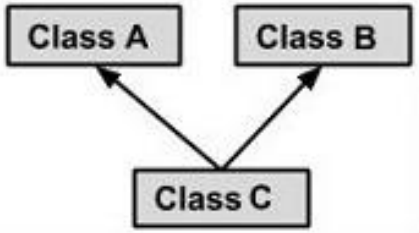
Keyword “instanceof”

```
class Animal {}
class Mammal extends Animal {}

public class Dog extends Mammal {

    public static void main(String args[]) {
        Mammal m = new Mammal();
        Dog d = new Dog();

        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal);
    }
}
```

Types of Inheritance

Single Inheritance
 
	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }
Hierarchical Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** A {..... }

Multiple Inheritance
 
	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

Overriding

- Overriding can be prevented by using modifier “final”

```
class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}
class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}
public class TestDog {
    public static void main(String args[]) {
        Animal a = new Animal();   // Animal reference and object
        Animal b = new Dog();   // Animal reference but Dog object

        a.move();   // runs the method in Animal class
        b.move();   // runs the method in Dog class
    }
}
```

Java Abstraction

- Define the functionality but hide the details of implementation
- Abstract Class
  - if a class has at least one abstract method, then the class **must** be abstract
  - If a class is declared abstract, it cannot be instantiated
  - To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- Abstract Method

```
public abstract class Shape {
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
    public abstract int area();

    int width;
    int height;
};
```

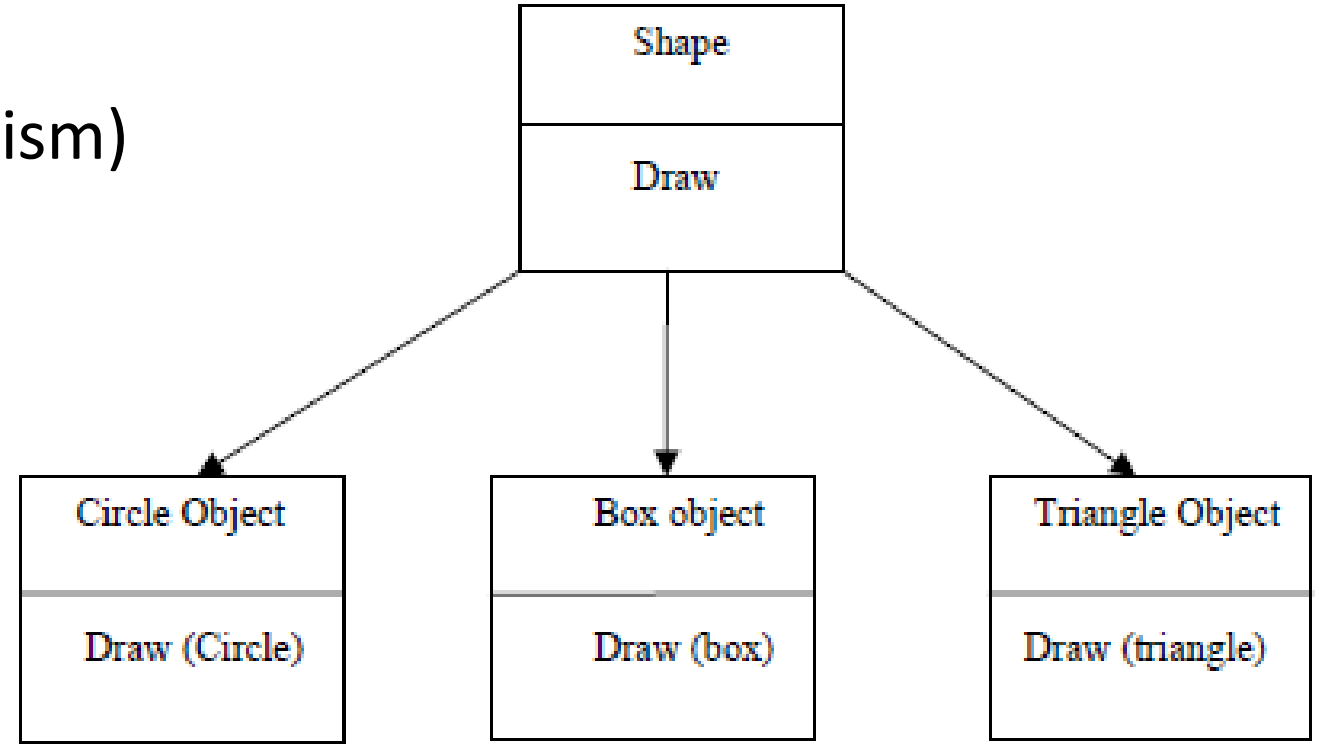
Java Encapsulation

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

```
class Student {
    private String name;
    private String id;
    private int age;

    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getId() {
        return id;
    }
    public void setAge(int newAge) {
        age = newAge;
    }
    public void setName(String newName) {
        name = newName;
    }
    public void setId(String newId) {
        id = newId;
    }
}
```

Polymorphism

- Overriding (Dynamic Polymorphism)
- Overloading (Static Polymorphism)



Java Interface

- Interface is implcitly abstract
- All the methods in an interface are abstract
- An interface cannot contain instance fields, only static and final field
- Interface can extend multiple interfaces

```
interface Animal {
    public void eat();
    public void travel();
}

public class MammalInt implements Animal {
    public void eat() {
        System.out.println("Mammal eats");
    }
    public void travel() {
        System.out.println("Mammal travels");
    }
    public static void main(String args[]) {
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}
```

Java Interface

- Extending Multiple Interfaces

```
public interface Hockey extends Sports, Event
```

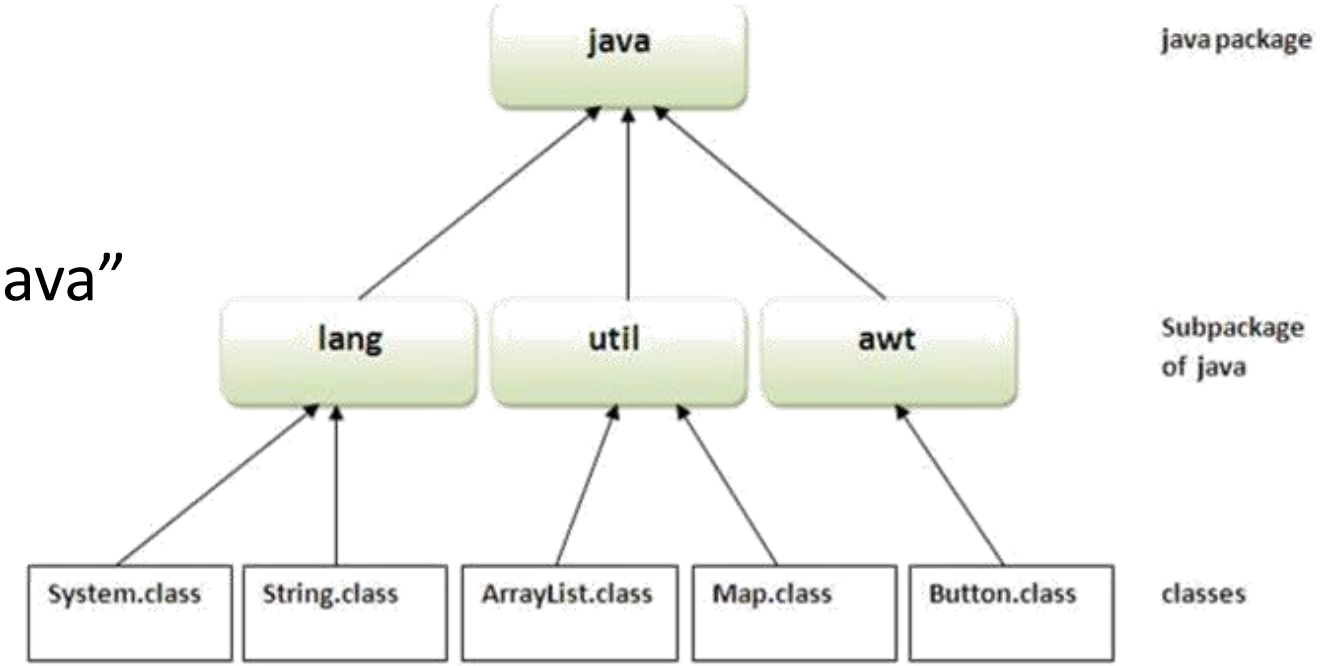
- Tagging Interfaces

```
package java.util;
public interface EventListener {}
```

Java Packages

- Prevent name conflicts
- Each package maps to a folder
  - .\com\apple\computers\Dell.java
- Use keyword “package” in “Dell.java”

```
package com.apple.computers;
public class Dell {
…
}
…
```



- Compile a package
  - javac -d destination_folder file_name.java
- Useage
  - import java.lang.String
  - import java.lang.*

References

- https://www.tutorialspoint.com/java/
- What is Java Virtual Machine?
- https://www.softwaretestingmaterial.com/operators-in-java/