JAVA 101

Piya Lumyong

Environment

SDKMAN!

- Tool for managing parallel versions of multiple Software Development Kits

- Inspired by RVM and rbenv

Install

```
curl -s "https://get.sdkman.io" | bash
```

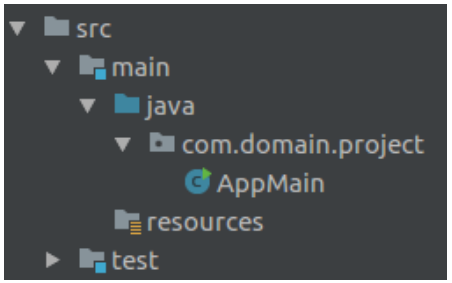
Install SDK

```
sdk install java
sdk install gradle
```

Reserved word

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

Folder structure & package

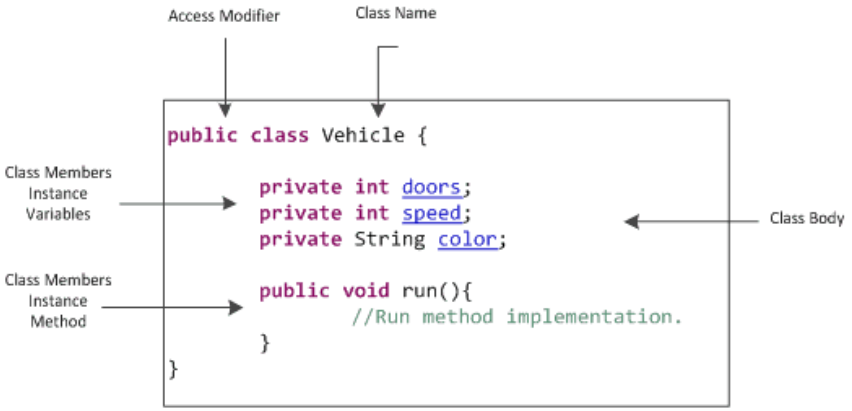



```java
package com.domain.project;

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

    }
}
```

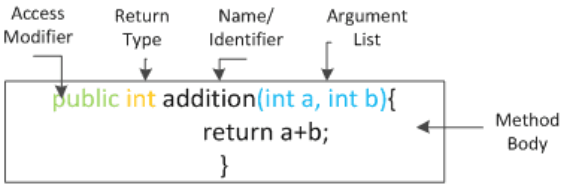
Package by features, not layers

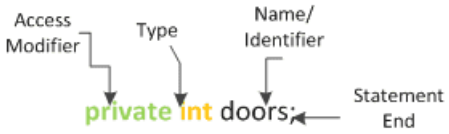
Structure of Java Code

Class



Method





Variable

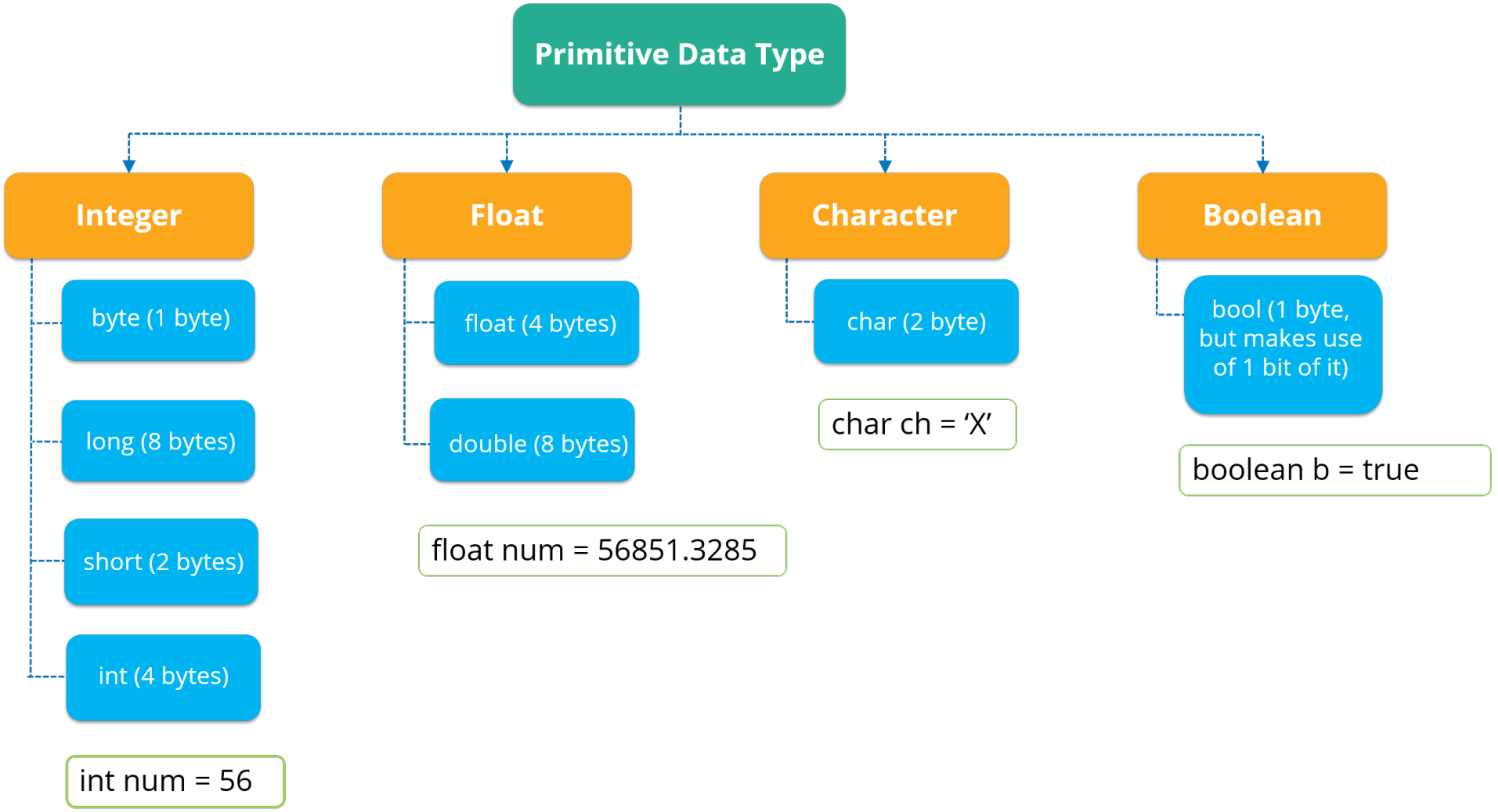
Type

Two Type

1 Primitives Type

2 Reference Type

Primitives Type



Ref. Edureka

Wrapper and Auto Boxing

Java Primitive Data Type	Wrapper Class
int	Integer
double	Double
boolean	Boolean
byte	Byte
char	Character
float	Float
long	Long
short	Short

```
long a = Long.parseLong("10");
Long b = Long.valueOf("10");
Long c = new Long(10);
long d = 10L;
Long e = 10L;
```

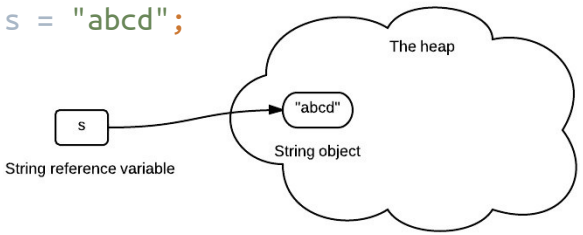
Array

```
int[ ] aryNums = new int[6];
aryNums[0] = 10;
aryNums[1] = 14;
aryNums[2] = 36;

int[ ] aryNums = new int[] { 1, 2, 3, 4 };
int[ ] aryNums = { 1, 2, 3, 4 };

String[ ] aryStrings = {"Autumn", "Spring", "Summer", "Winter" };


int[ ][ ] aryNumbers = new int[6][5];
aryNumbers[0][0] = 10;
aryNumbers[1][1] = 12;
aryNumbers[0][2] = 43;
aryNumbers[3][3] = 11;
AryNumbers[0][4] = 22;

String[][] matrix = {{"1:1", "1:2"}, {"2:1", "2:2"}};
```

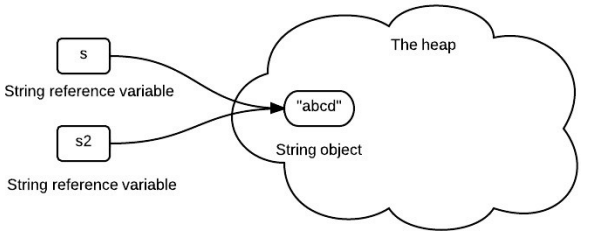
String

String is immutable!!!

```
String s = "abcd";
```

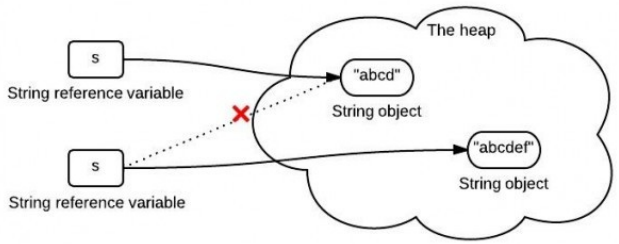



```
String s2 = "abcd";
```



String Pool

```
s = s.concat("ef");
Or
s = s + "ef";
```



String method

public class String		
	String(String s)	*create a string with the same value as* `s`
int	length()	*number of characters*
char	charAt(int i)	*the character at index* `i`
String	substring(int i, int j)	*characters at indices* `i` *through* `(j-1)`
boolean	contains(String substring)	*does this string contain* `substring` *?*
boolean	startsWith(String pre)	*does this string start with* `pre` *?*
boolean	endsWith(String post)	*does this string end with* `post` *?*
int	indexOf(String pattern)	*index of first occurrence of* `pattern`
int	indexOf(String pattern, int i)	*index of first occurrence of* `pattern` *after* `i`
String	concat(String t)	*this string with* `t` *appended*
int	compareTo(String t)	*string comparison*
String	toLowerCase()	*this string, with lowercase letters*
String	toUpperCase()	*this string, with uppercase letters*
String	replaceAll(String a, String b)	*this string, with* a *s replaced by* b *s*
String[]	split(String delimiter)	*strings between occurrences of* `delimiter`
boolean	equals(Object t)	*is this string's value the same as* `t`*'s ?*
int	hashCode()	*an integer hash code*

Formating String

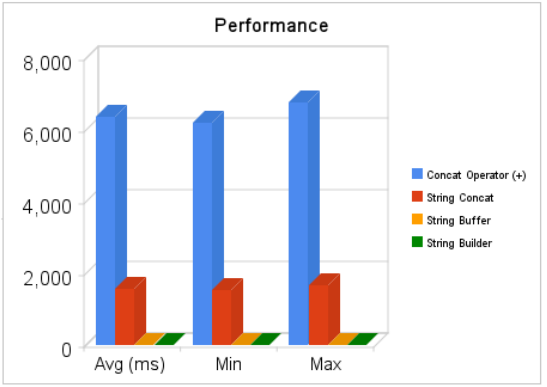
```java
String heading1 = "Exam_Name";
String heading2 = "Exam_Grade";
System.out.printf("%-15s %15s %n", heading1, heading2);

double price = 80;
int vat = 7;
System.out.printf("Total : %10.2f %nVat %d%%: %10.2f", price, vat, (price * vat / 100));

String.format("Total : %10.2f %nVat %d%%: %10.2f", price, vat, (price * vat / 100));
new Formatter().format("Total : %10.2f %nVat %d%%: %10.2f", price, vat, (price * vat / 100));
```

```
/media/src/ide/tool/sdkman/candidates/java/current/bin/java ...
Exam_Name            Exam_Grade
JAVA                      A
PHP                       B
.NET                      A

Process finished with exit code 0
```

StringBuilder & StringBuffer

StringBuffer	StringBuilder
Thread safe	Not thread safe
Synchronized	Not synchronized
Slower than StringBuilder	Faster than StringBuffer

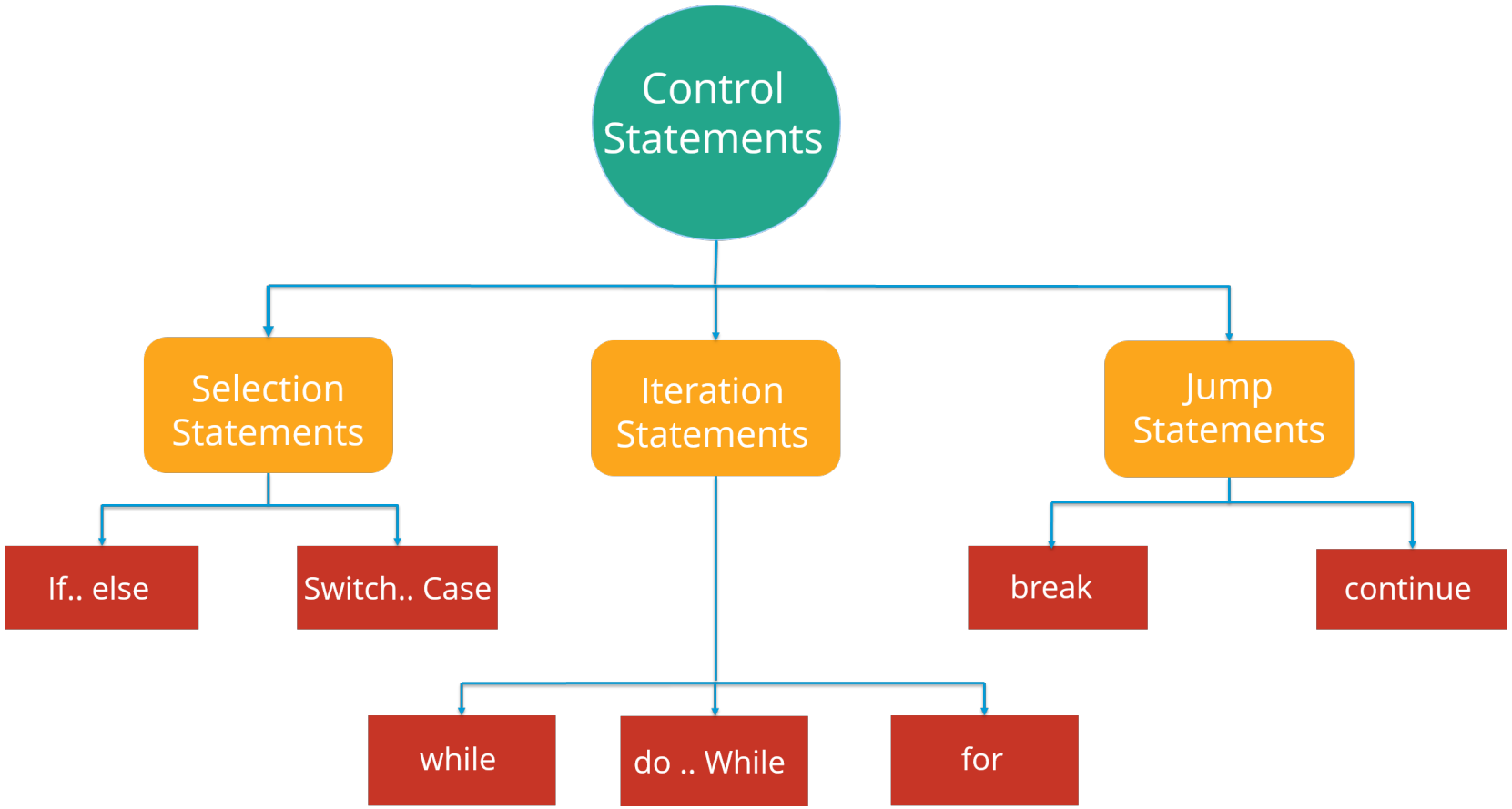
	Avg (ms)	Min	Max	Std Dev
Concat Operator (+)	6357.3	6199	6770	162.6
String Concat	1582.1	1527	1659	38.3
String Buffer	2.8	1	5	1.2
String Builder	2.4	1	4	1




```
StringBuilder sb = new StringBuilder("Hello...");

char c = '!';
sb.append(c);

sb.insert(8, " Java");

sb.delete(5,8);

System.out.println(sb);
```

Ref. Performance

Flow Control



Ref. Edureka

Flow Control - If

```
if (condition_one) {
} else if (condition_two) {
} else {
}
```

Flow Control - Boolean Value

> Greater Than
< Less Than
>= Greater Than or Equal To
<= Less Than or Equal To

& AND
| OR
&& Short AND
|| Short OR
== HAS A VALUE OF
! NOT

Flow Control - If

```
if (user <= 18) {
    System.out.println("User is younger");
} else if (user > 18 && user <= 35) {
    System.out.println("User is adults");
} else if (user > 35 && user <= 55) {
    System.out.println("User is middle-age");
} else {
    System.out.println("User is older");
}
```

Flow Control – Short If

```
Type variable = (condition) ? value1_of_type : value2_of_type
```

```
interval = interval < 30 ? 30 : interval;
boolean admin = userType.equals("Admin") ? true : false;
```

Flow Control – Switch Case

```
switch ( variable_to_test ) {
  case value_constant:
    code_here;
    break;
  case value_constant:
    code_here;
    break;
  default:
    values_not_caught_above;
}
```

Flow Control - For

```
for (start_value; end_value; increment_number) {
   //YOUR_CODE_HERE
}
```

```
String[] fruits = {"banana", "apple", "coconut"};
for (int i = 0; i < fruits.length; i++) {
  String fruit = fruits[i];

  //YOUR_CODE_HERE
}
```

```
for (String fruit : fruits) {
   //YOUR_CODE_HERE
}
```

Flow Control - While

```
while (condition) {
      //YOUR_CODE_HERE
}

do {
      //YOUR_CODE_HERE
} while (condition)
```

```
try (FileReader fr = new FileReader("name.txt")) {
   BufferedReader br = new BufferedReader(fr);
   String line;
   while ((line = br.readLine()) != null) {
      System.out.println(line);
   }
} catch (IOException e) {
   e.printStackTrace();
}
```

Exercise Min Max Avg

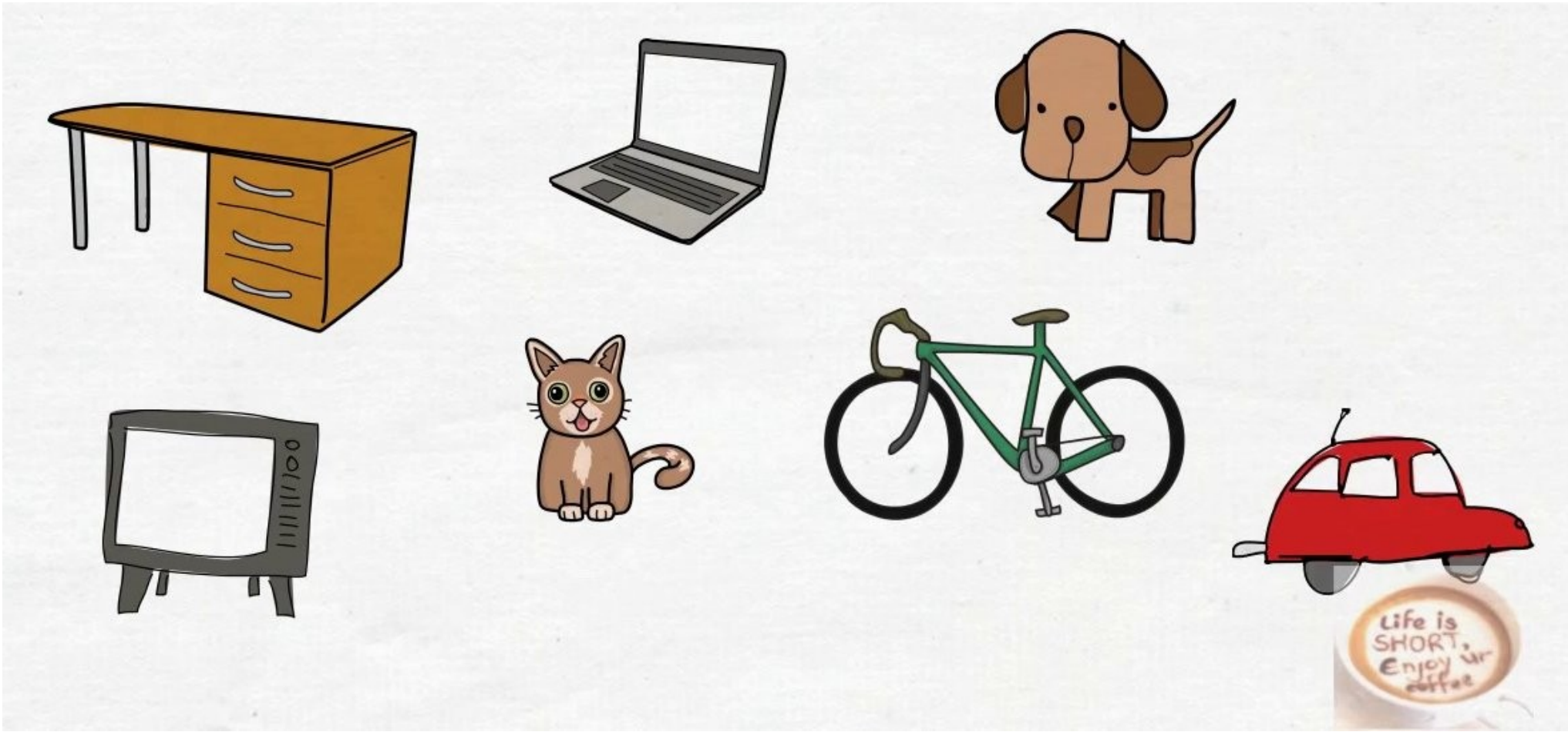
```
String[] valules = {"43", "352", "32", "79", "578", "54"};
```

Exercise - Factorial

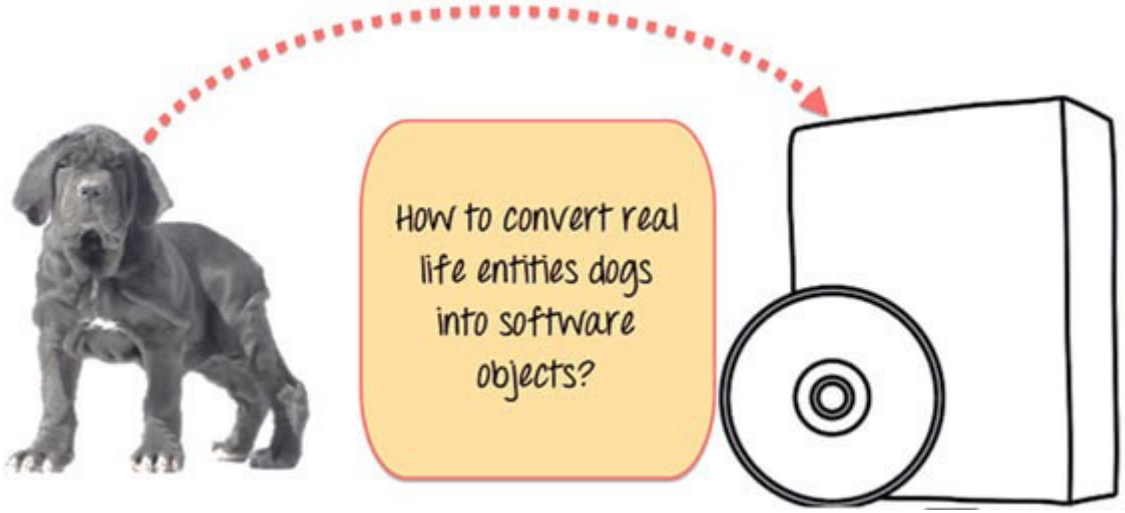
$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

n	***n*!**
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5 040

Class

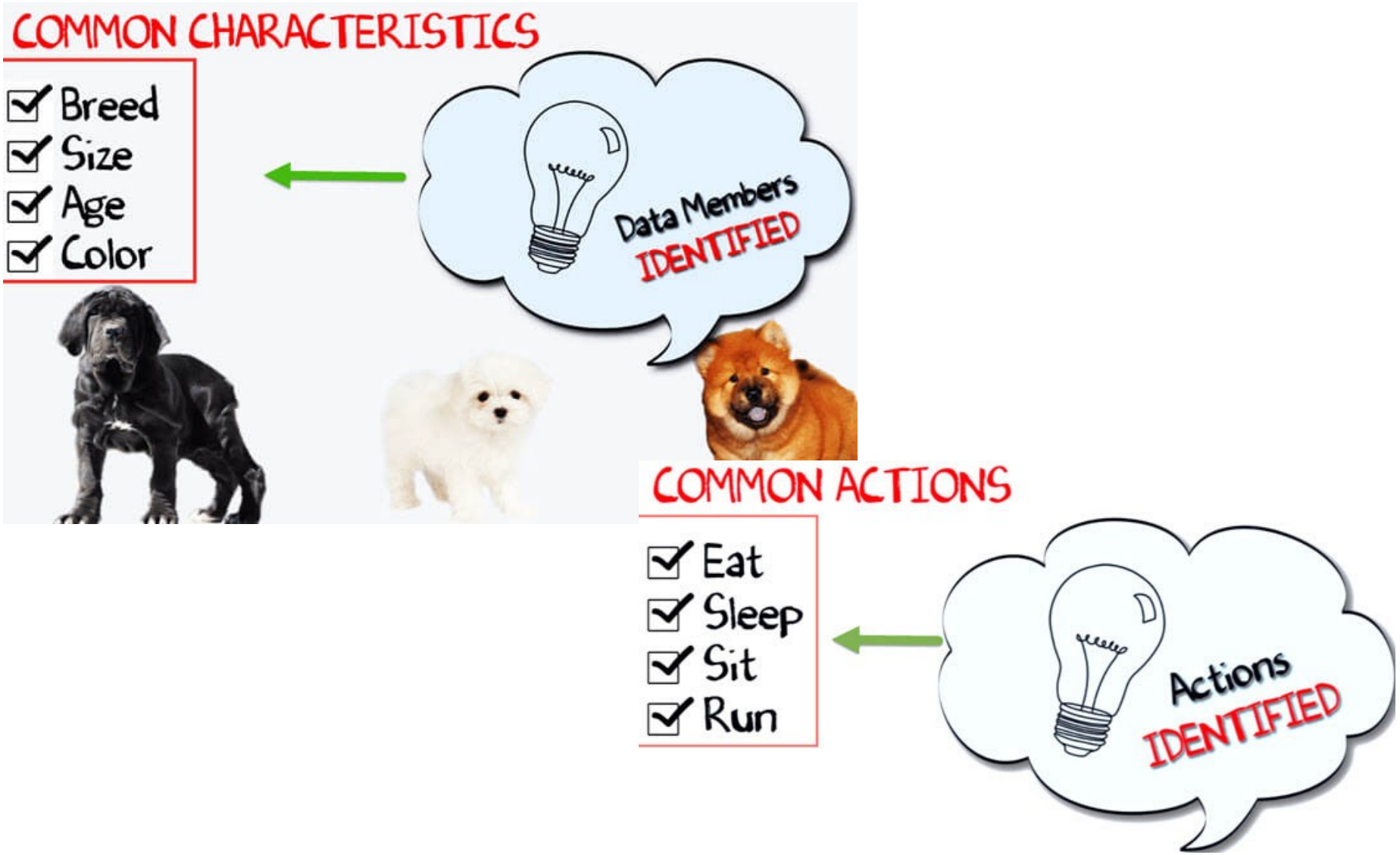


Class



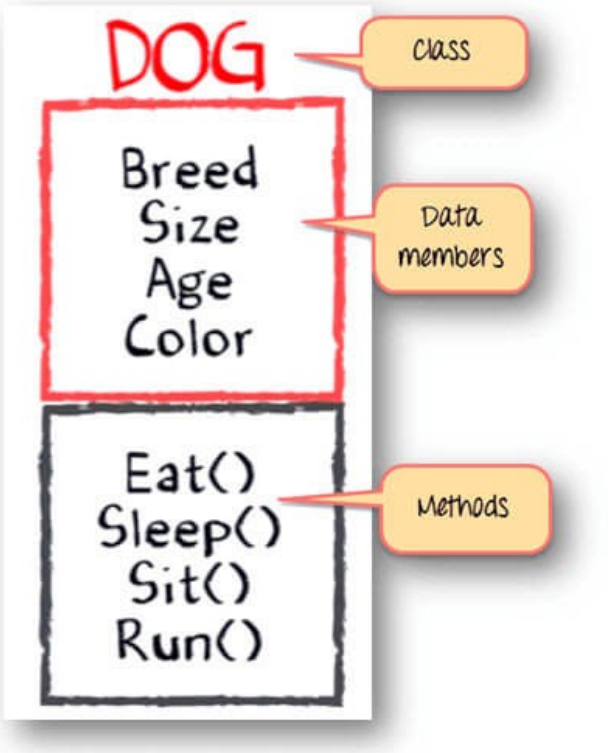
Ref. Guru99

Class



Ref. Guru99

Class



Ref. Guru99

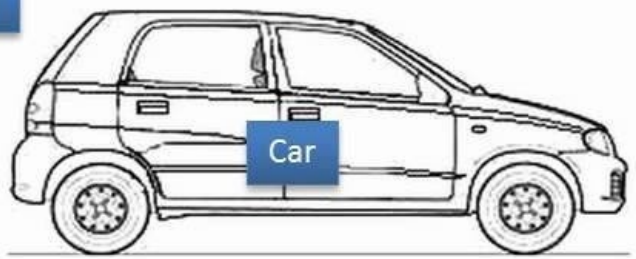
Class vs Instance

Class vs Instance

What is a Class?

A *class* is the blueprint from which individual objects are created.

Class




```java
public class Car
{
    private String brand = null;
    private String model = null;
    private String color = null;

    public String getBrand()
    {
        return brand;
    }

    public void setBrand(String brand)
    {
        this.brand = brand;
    }

    public String getModel()
    {
        return model;
    }

    public void setModel(String model)
    {
        this.model = model;
    }

    public String getColor()
    {
        return color;
    }

    public void setColor(String color)
    {
        this.color = color;
    }
}
```

Objects

```java
Car maruthiAltok10 = new Car();

maruthiAltok10.setBrand("Maruthi Alto");
maruthiAltok10.setModel("k10");
maruthiAltok10.setColor("Orange");
```

brand = Maruthi Alto
model = K10
color = Orange

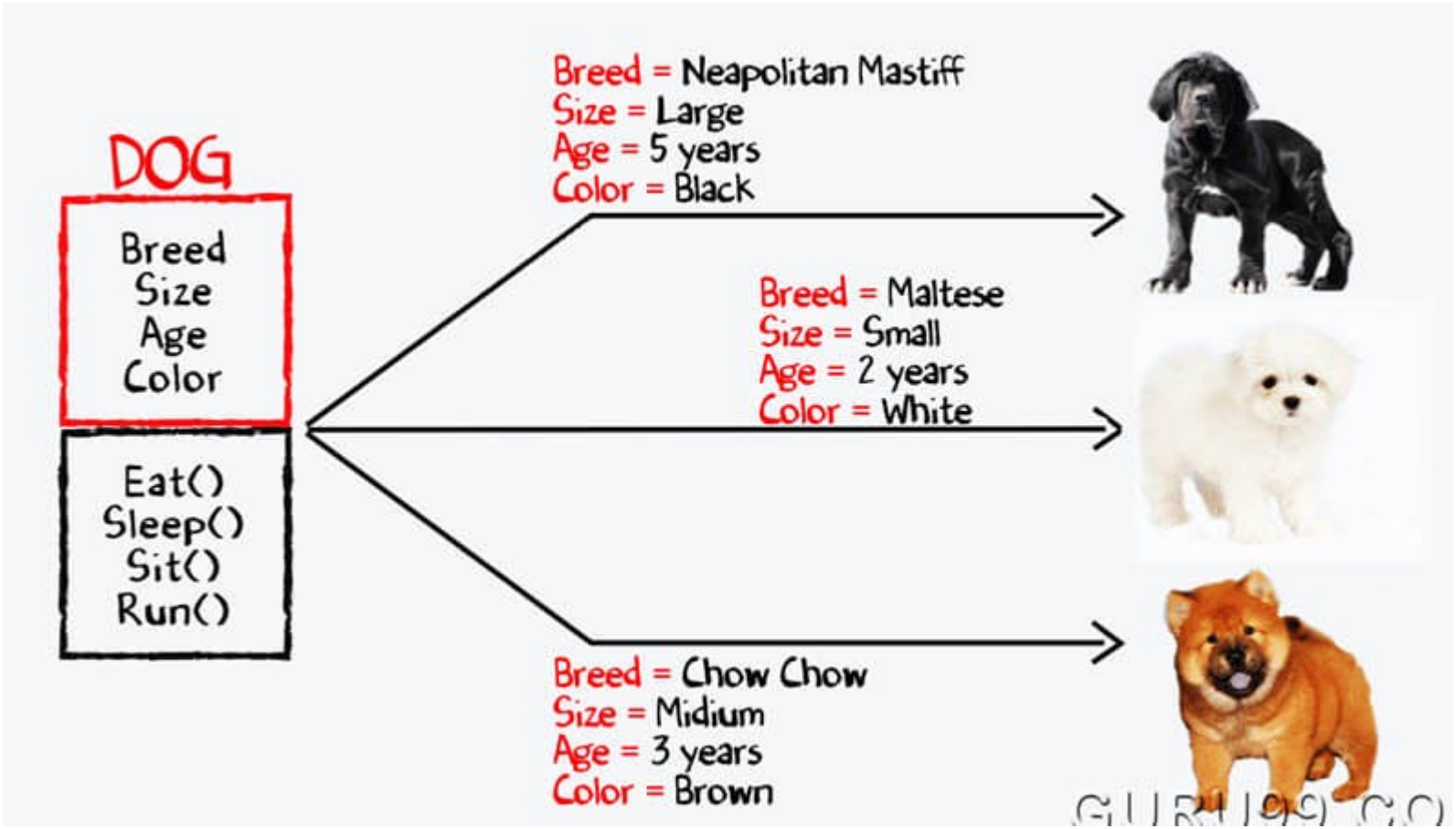
```java
Car swift = new Car();

swift.setBrand("Swift");
swift.setModel("ZDI");
swift.setColor("Red");
```

brand = Swift
model = ZDI
color = Red

```java
Car maruthiAlto800 = new Car();

maruthiAlto800.setBrand("Maruthi Alto");
maruthiAlto800.setModel("800");
maruthiAlto800.setColor("Blue");
```

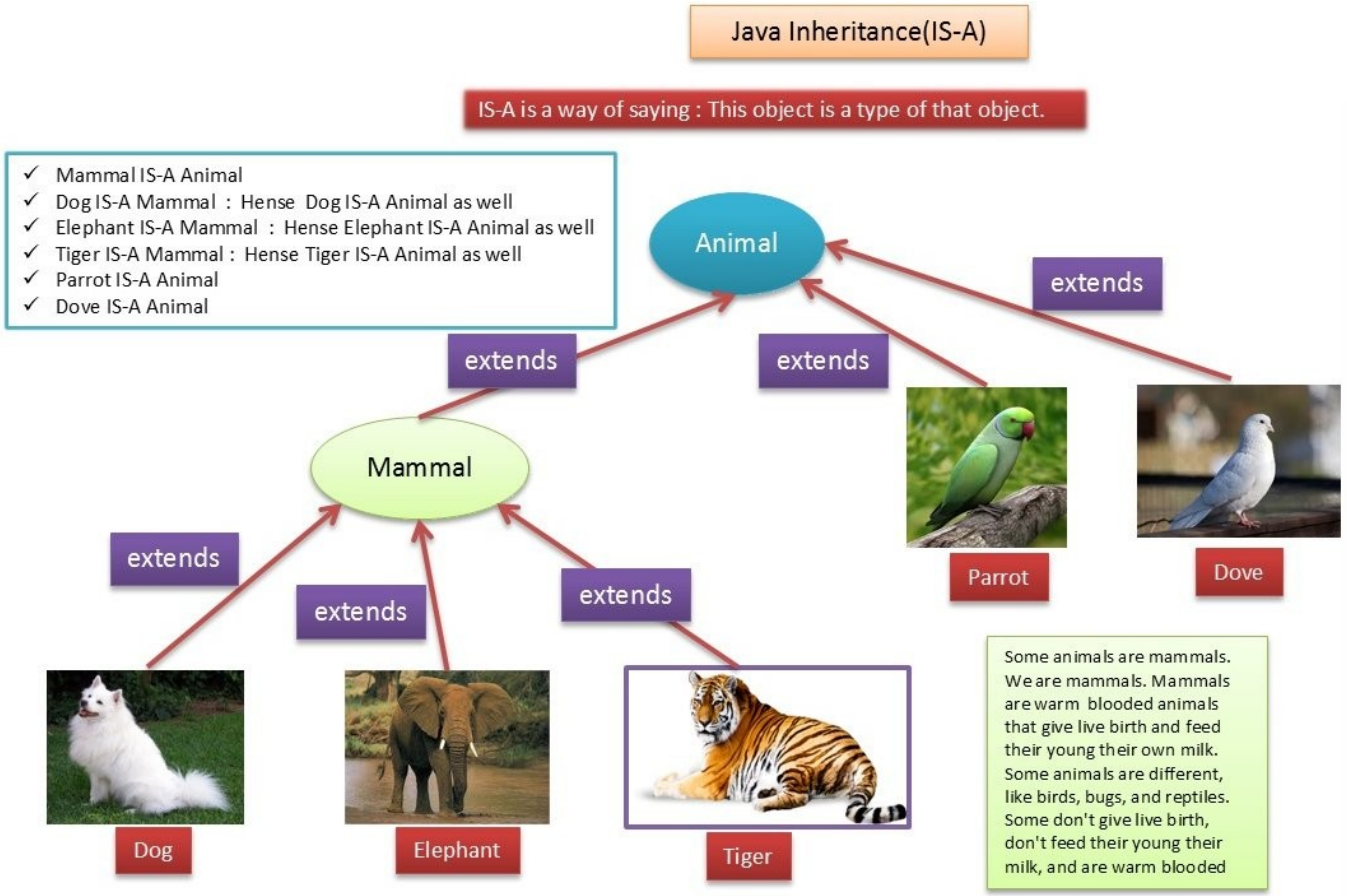
brand = Maruthi Alto
model = 800
color = Blue

Class vs Instance



Ref. Guru99

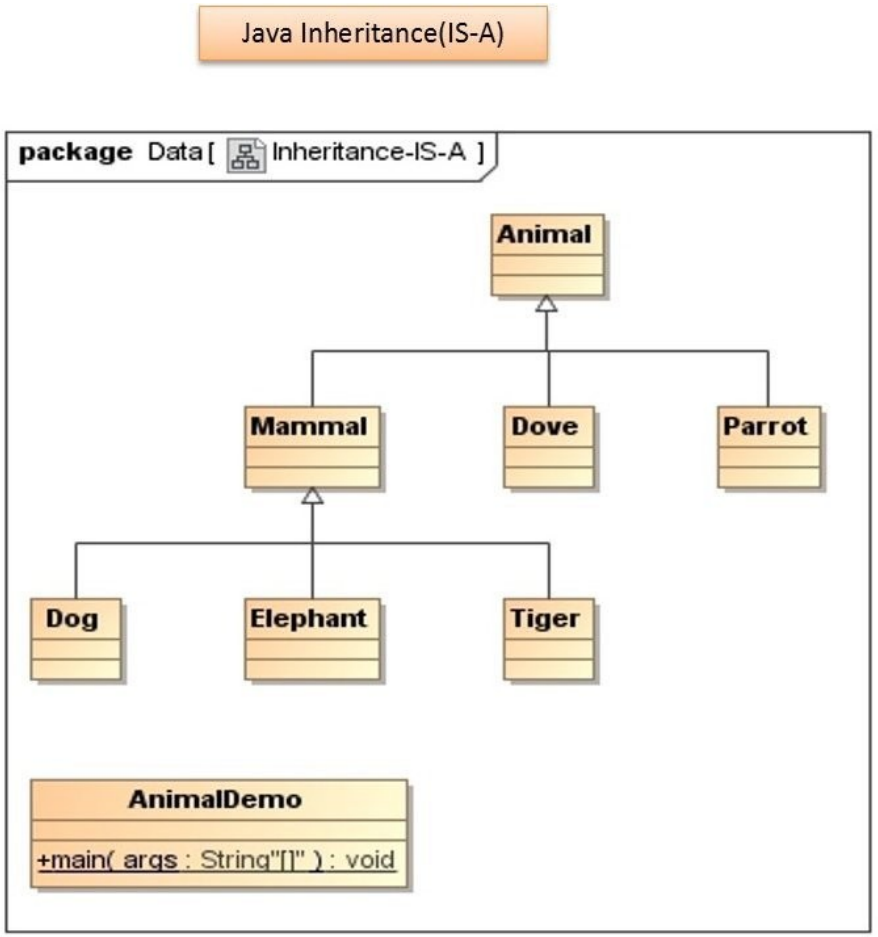
Inheritance




```
public class Animal
{

}

public class Mammal extends Animal
{

}

public class Dog extends Mammal
{

}

public class Elephant extends Mammal
{

}

public class Tiger extends Mammal
{

}

public class Parrot extends Animal
{

}

public class Dove extends Animal
{

}
```

Ref. ramj2ee

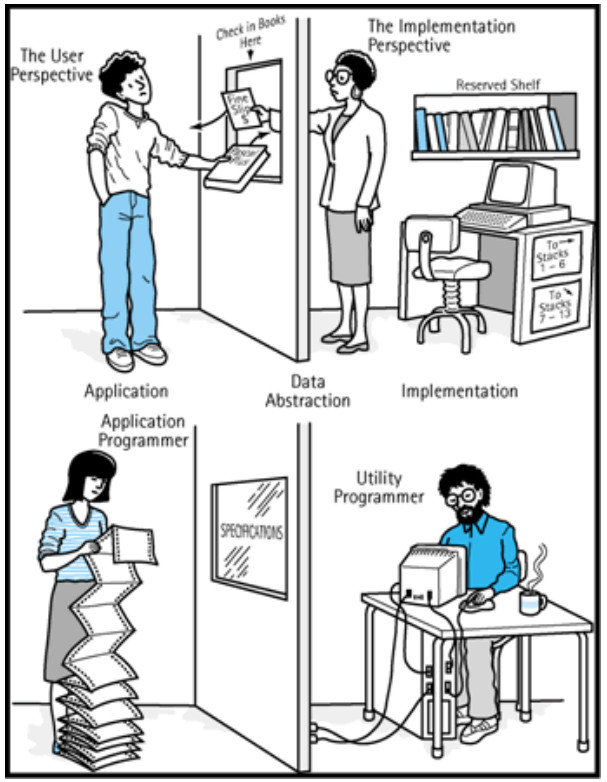
Inherite



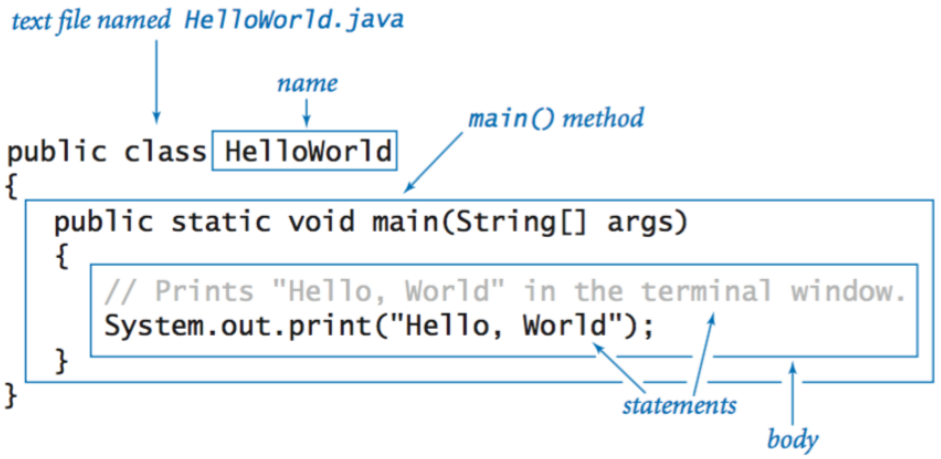
Ref. ramj2ee

Definition of OOP Concepts

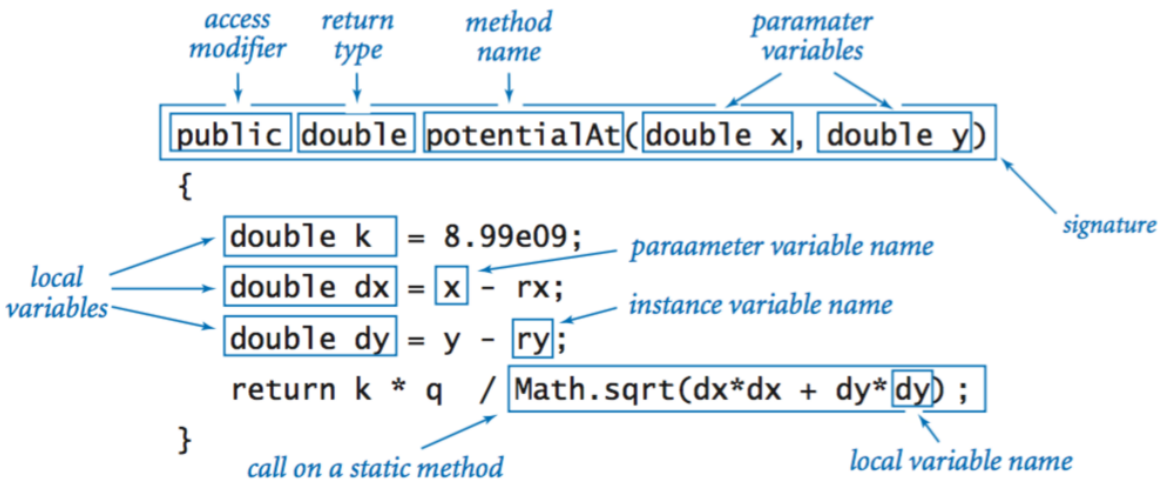
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism



Class


text file named HelloWorld.java
name
main() method
public class HelloWorld
{
public static void main(String[] args)
{
// Prints "Hello, World" in the terminal window.
System.out.print("Hello, World");
}
}
statements
body

Method


access modifier
return type
method name
paramater variables
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy) ;
}
signature
local variables
paraameter variable name
instance variable name
call on a static method
local variable name

Constructor

```
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 1, 1);
    }

    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }

    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

Inheritance

```
/**
 * Animal is the Super class of Mammal,Parrot and Dove classes
 */
class Animal
{

}
```

```
        /**
         * Parrot is the subclass of Animal classes.
         */
        class Parrot extends Animal
        {

        }

        /**
         * Dove is the subclass of Animal classes.
         */
        class Dove extends Animal
        {

        }
```

```
/**
 * Mammal is the subclass of Animal class.
 */
class Mammal extends Animal
{

}

/**
 * Dog is the subclass of both Mammal and Animal classes.
 */
class Dog extends Mammal
{

}

/**
 * Elephant is the subclass of both Mammal and Animal classes.
 */
class Elephant extends Mammal
{

}

/**
 * Tiger is the subclass of both Mammal and Animal classes.
 */
class Tiger extends Mammal
{

}
```

Interface

```
// I say all motor vehicles should look like this:
interface MotorVehicle
{
    void run();

    int getFuel();
}

// My team mate complies and writes vehicle looking that way
class Car implements MotorVehicle
{

    int fuel;

    void run()
    {
        print("Wrroooooooom");
    }


    int getFuel()
    {
        return this.fuel;
    }
}
```

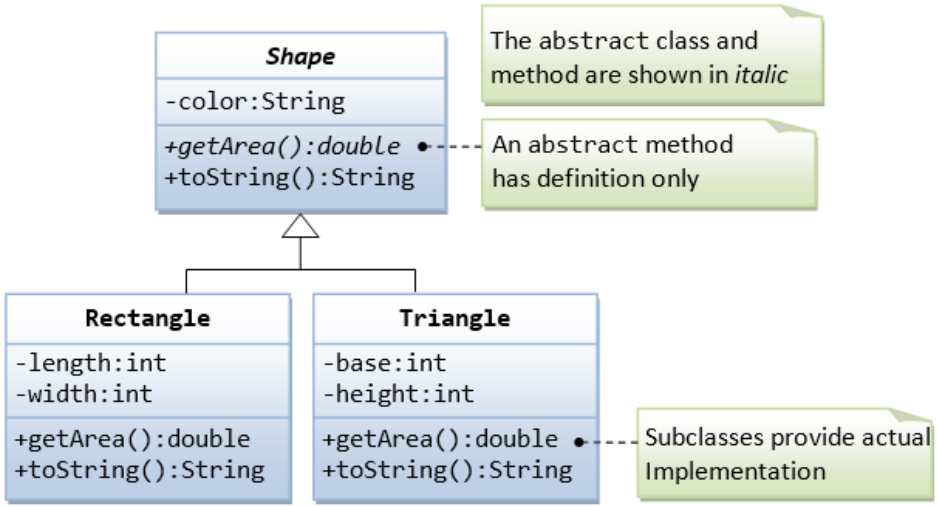
Abstract

```
// I say all motor vehicles should look like this:
abstract class MotorVehicle
{

    int fuel;

    // They ALL have fuel, so lets implement this for everybody.
    int getFuel()
    {
         return this.fuel;
    }

    // That can be very different, force them to provide their
    // own implementation.
    abstract void run();
}

// My teammate complies and writes vehicle looking that way
class Car extends MotorVehicle
{
    void run()
    {
        print("Wrroooooooom");
    }
}
```

Comparison Abstract And Interface

Abstract Class	Interface
Abstract keyword is used to create an abstract class and it can be used with methods.	Interface keyword is used to create an interface but it cannot be used with methods.
A class can extend only one abstract class.	A class can implement more than one interface.
An abstract class can have both abstract and non-abstract methods.	An interface can have only abstract methods.
Variables are not final by default. It may contain non-final variables.	Variables are final by default in an interface.
An abstract class can provide the implementation of an interface.	An interface cannot provide the implementation of an abstract class.
It can have methods with implementations.	It provides absolute abstraction and cannot have method implementations.
It can have public, private, static and protected access modifiers.	The methods are implicitly public and abstract in Java interface.
It doesn’t support multiple inheritances.	It supports multiple inheritances.
It is ideal for code reuse and evolution perspective.	It is ideal for Type declaration.

Exercise




```java
/*
 * Superclass Shape maintain the common properties of all shapes
 */
public class Shape {
   // Private member variable
   private String color;
   // Constructor
   public Shape (String color) {
      this.color = color;
   }
   @Override
   public String toString() {
      return "Shape[color=" + color + "]";
   }
   // All shapes must have a method called getArea().
   public double getArea() {
      // We have a problem here!
      // We need to return some value to compile the program.
      System.err.println("Shape unknown! Cannot compute area!");
      return 0;
   }
}
```

Access Modifier

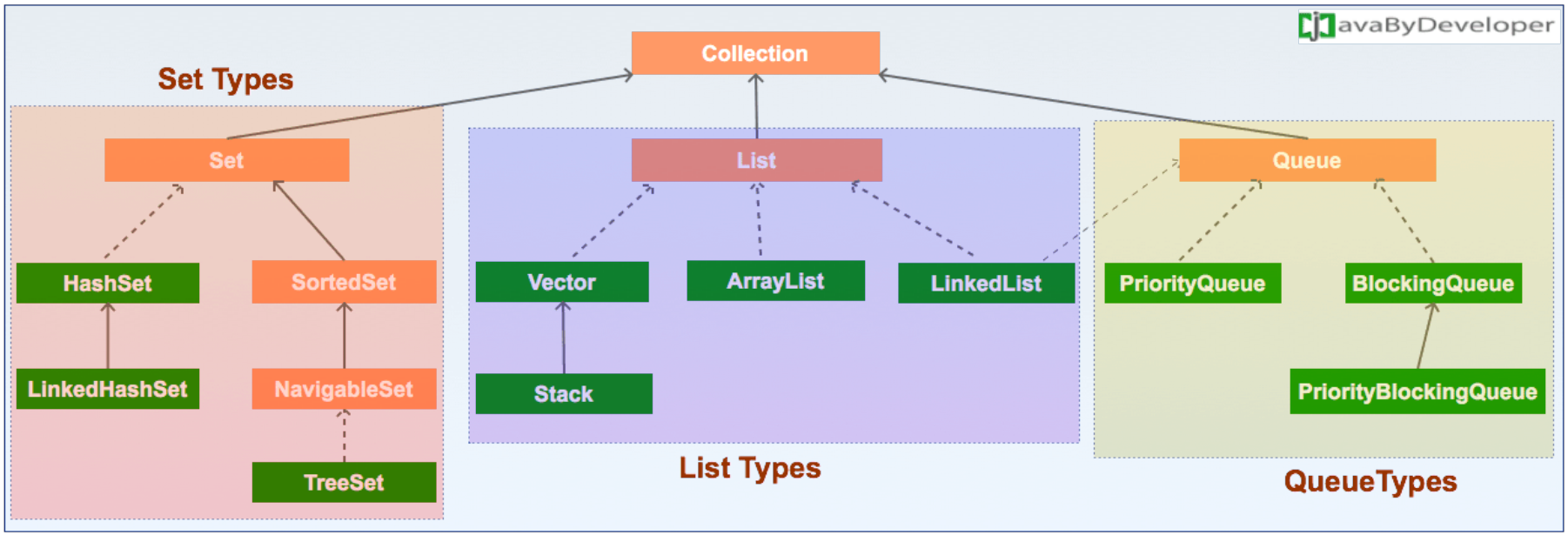
Modifier	class	constructor	method	Data/variables
public	Yes	Yes	Yes	Yes
protected	Yes*	Yes	Yes	Yes
default	Yes	Yes	Yes	Yes
private	Yes*	Yes	Yes	Yes
static	Yes*		Yes	Yes*
final	Yes		Yes	Yes

Access Level

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible
blank : not accessible

Collection



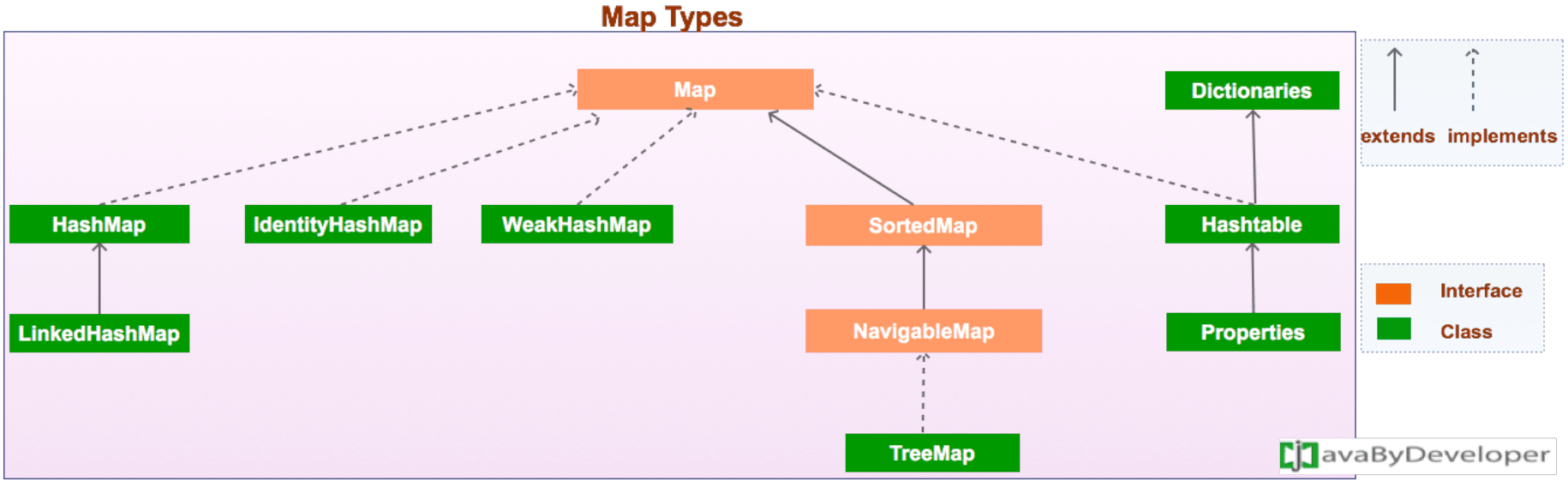
Insert order not preserve

Order

Priority

Ref. JavaByDeveloper

Java Collection



Ref. JavaByDeveloper

Comparator & Comparable

```java
import java.util.Comparator;

class StudentIDComparator implements Comparator<Student>{
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getId().compareTo(o2.getId());
    }
}

class StudentNameComparator implements Comparator<Student>{
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getName().compareTo(o2.getName());
    }
}
```

```java
public class Student implements Comparable<Student>{
    private String id;
    private String name;
    private int score;

    Student(String id, String name, int score){
        this.id = id;
        this.name = name;
        this.score = score;
    }

    @Override
    public int compareTo(Student o) {
        return this.id.compareTo(o.id);
    }

    public String toString(){
        return "\n id: "+id+" name: "+name +" score: "+score;
    }
}
```

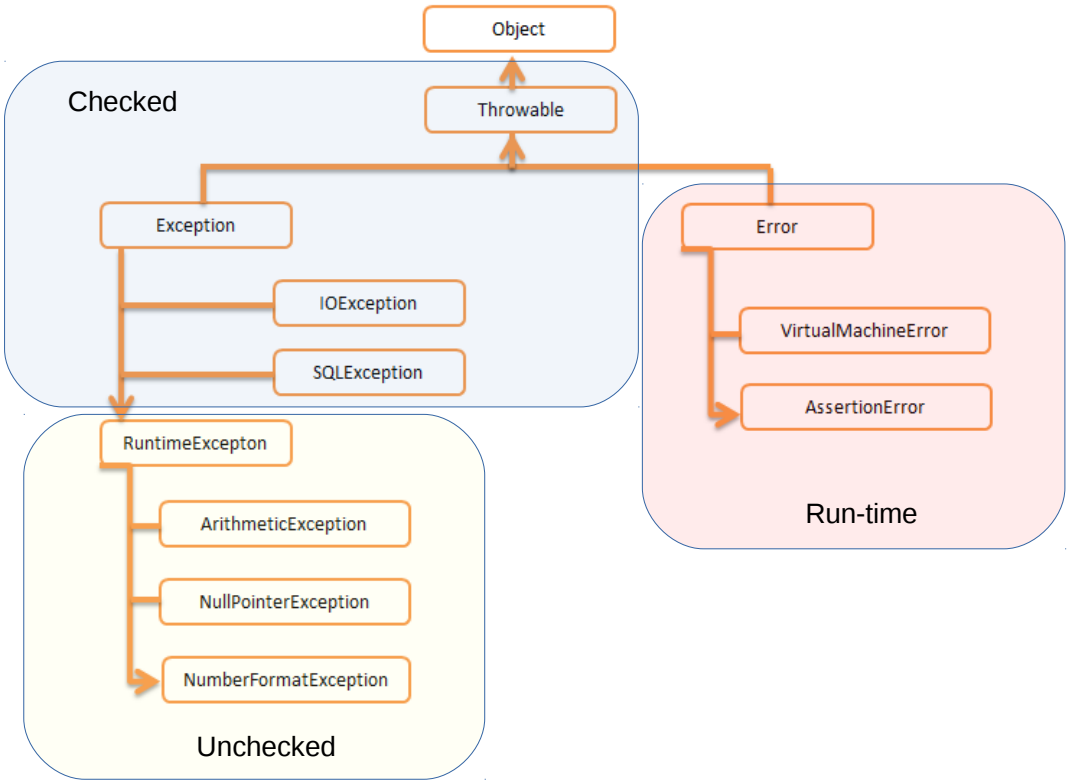
Comparison Comparable & Comparable

java.lang.Comparable	java.util.Comparator
The method in the Comparable interface is declared as int compareTo(ClassType type);	The method in the Comparator interface is declared as int compare(ClassType type1, ClassType type2);
Returns negative if objOne < objTwo zero if objOne == objTwo positive if objOne > objTwo	Same as Comparable
You must modify the class whose instances you want to sort.	You build a class separate from the class whose instances you want to sort.
Implemented frequently in the API by: String, Wrapper classes, Date, Calendar…	Meant to be implemented to sort instances of third-party classes.
Used when the objects need to be compared in their natural order.	Used when the objects need to be compared in custom user-defined order (other than the natural order).
You do not create a separate class just to implement the Comparable interface	You create a separate class just to implement the Comparator interface.
For a given class type, you have only that class (and that class alone) implementing the Comparable interface.	You can have many separate (i.e., independent) classes implementing the Comparator interface, with each class defining different ways to compare objects.

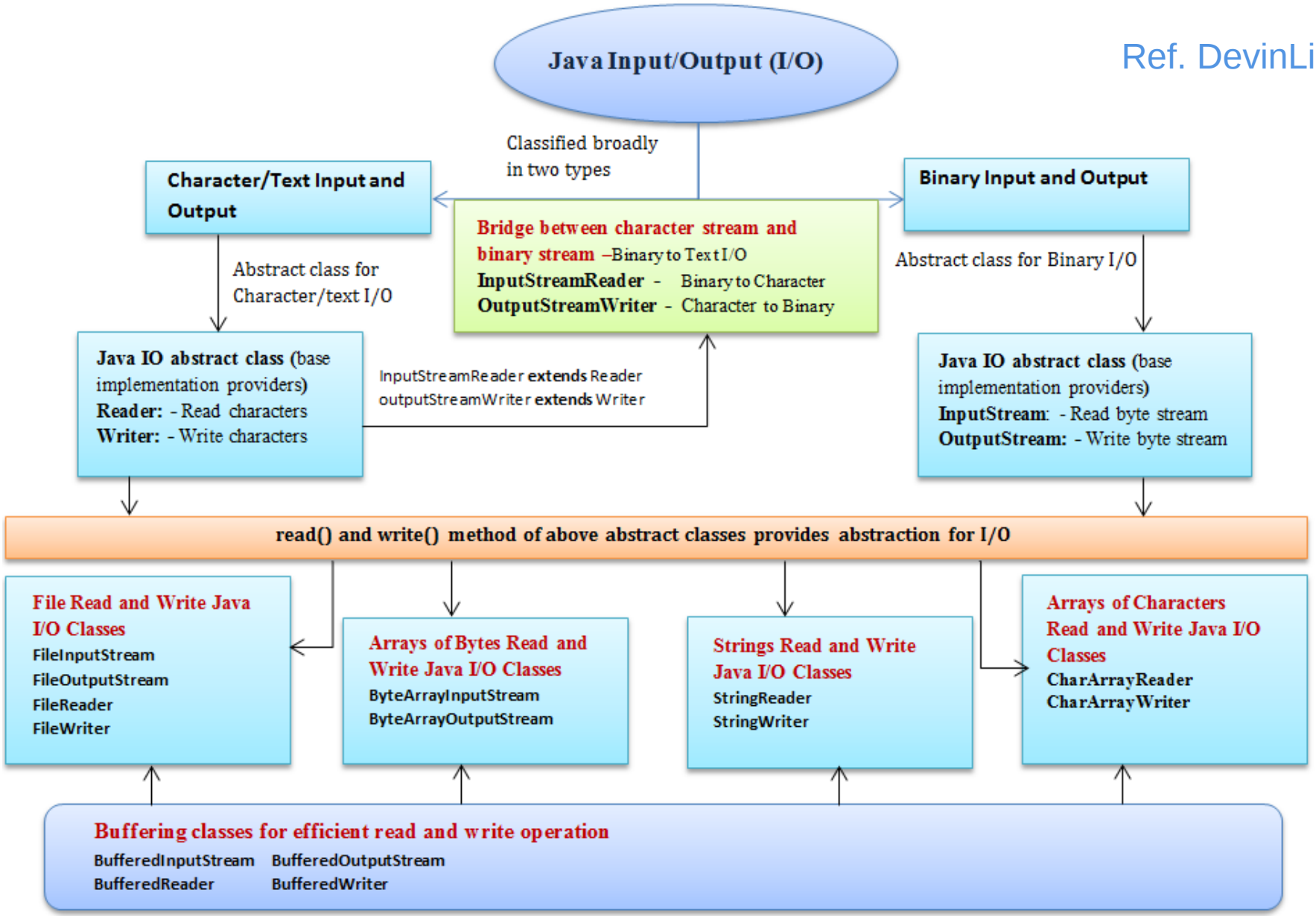
Exercise order employee in collection by name

```
class Employee {
    private String department;
    private String name;
    private int age;
    public Employee(String department, String name, int age) {
        this.name = name;
        this.department = department;
        this.age = age;
    }
    // Getter
    @Override
    public String toString() {
        return "Employee{" +
                "department='" + department + '\'' +
                ", name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Java Error Handling



Java I/O



Ref. DevinLine