Refresher : Core Java Concepts

Topics being covered :

1. OO concepts
2. Brief history about Java
3. Different types of Inheritance
4. Interfaces
5. Abstract Classes
6. Rules for declaring identifiers
7. What is JVM ?

This Module is for recalling all Necessary Core Java concepts, before starting android Programming

---

This Module is **not** asked in the Certification test

Class

Object

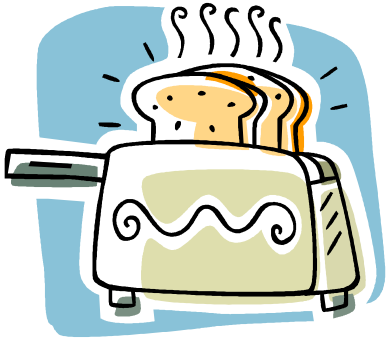
Class is a Template

Object is a copy of the Template

Real life Entities

- No. of windows
- door_locked
- Carpet area
- Pincode

- No. of seats
- No. of tires
- Engine cc
- License No.



- Watts
- Toaster type
- Make
- Warranty

Each Entity → Attributes

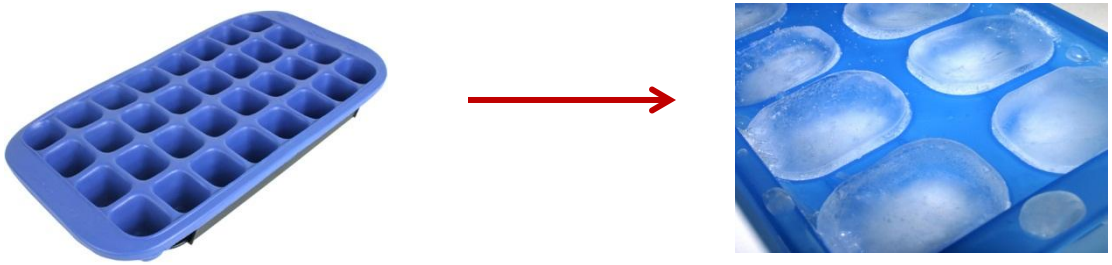
- getAddress ()
- isLocked()
- driving ()
- parking()

And methods too

- toast()

Entity → Class : is defined as a collection of Attributes(properties) and methods which operate on those attributes.

Class does not consume space



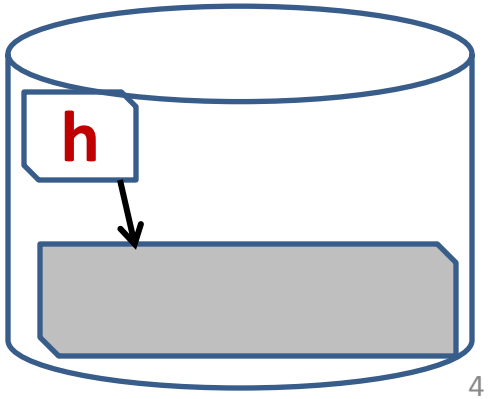
Object consumes space [Equal to total space taken by all attributes]

Object is defined as an Instance of class

In Java we make Objects in 2 steps

Declaration `House h ;`

Instantiation `h = new House() ;`



A Class Example :

```
class Box {
  double width;
  double height;
  double depth;

// Initialize the data values for the Box.
void  setBox() {
    width = 10;          height = 10;          depth = 10;
  }

// compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

Make object of Box

```
Box mybox1 = new Box();

double vol;

// set attributes of the Box
mybox1.setBox();

// get volume of the box
vol = mybox1.volume();

System.out.println("Volume
is " + vol);
```

Related to Class and Object we have many OOP concept

Inheritance	**Derived class inheriting properties of Parent class**
Polymorphism	**An object behaving differently in different situations**
Encapsulation	**Putting the Data and methods together in a class**
Abstraction	**Using the data only via methods of the class**

Inheritance

In Java → we use extends k/w
E.g. : class sportsCar extends Vehicle

Vehicle → Generic class → Parent class or Super class

Vehicle Data
- int engine_cc
- int seating_capc
- float price
- int gear

Vehicle Action
- changeGear()
- getSpec()
- setSpec()

Sports Car → specialized class → child or sub class

Sports_Car Data
- int highest_speed

Sports_Car Action
- void driveFastMode()

The following classes inherit from ?

Person

Common properties or Data
name , gender , age , height, weight, profession

battingAvg

softwares

No_of_gifts

Polymorphism

An object behaving differently in different situations

---

float area(int l , int b)

float area(float radius)

float area(int side)

All method names are same but work differently

This is called Polymorphism

OR Method Overloading

Encapsulation

```
Class Point
{
 int x;
 int y;
 void getX() { System.out.println("X = "+x);}
 void getY() { System.out.println("Y = "+y);}
 void setX() { x = 3;  }
 void setY() { y= 5;  }
}
```

Putting the Data and methods together in a class

Data

Methods

Abstraction

Using the data only via methods of the class

Abstraction

We can access data of Point Class in 2 ways :

```
Point p ;
p = new Point();
p.x = 10;
p.y = 10;
```

```
Point p ;
p = new Point();
p.setX();
p.setY();
```

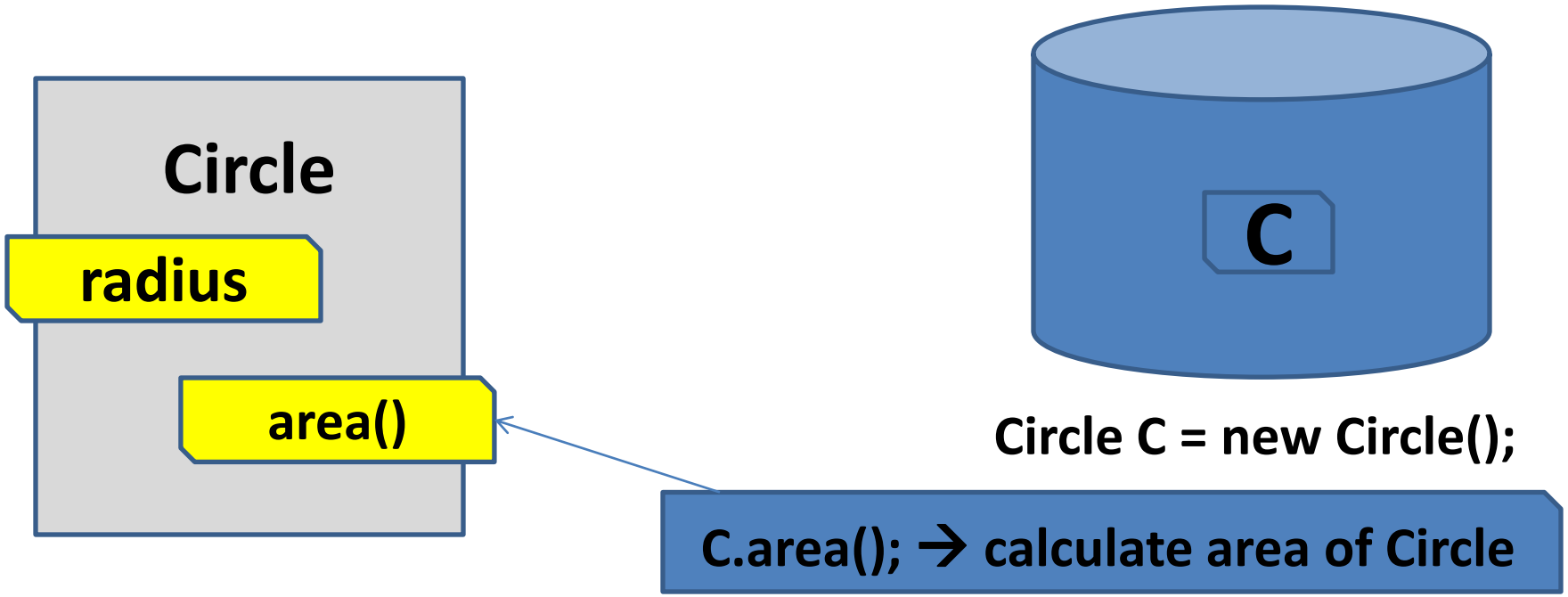
Ensures Abstraction

Advanced Concepts

To call method of a class , we use

Object_name.method_name()

Message Passing

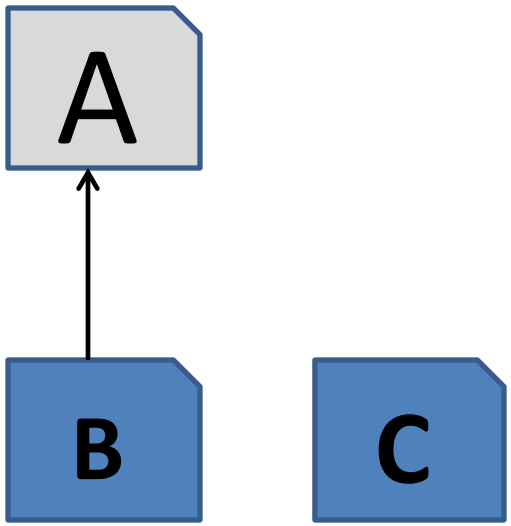


Pass a message (area) to an object C → means object C calls method area.

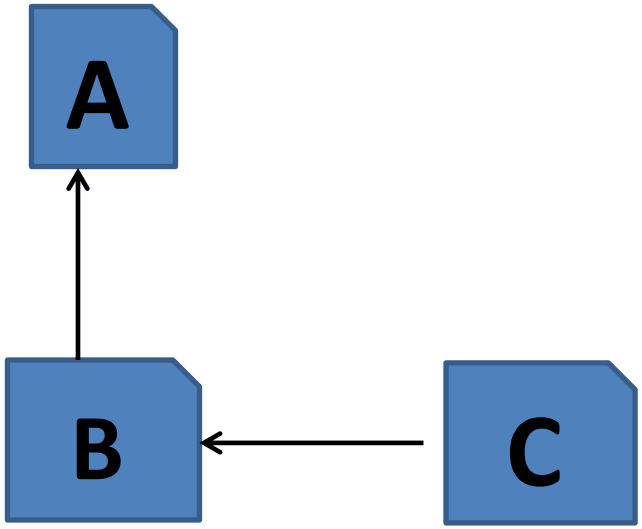
Coupling

Coupling is a measure of

“ how much is one class dependent upon others ”



Case 1: make changes to A



Case 2: make changes to A
How many classes get affected ?

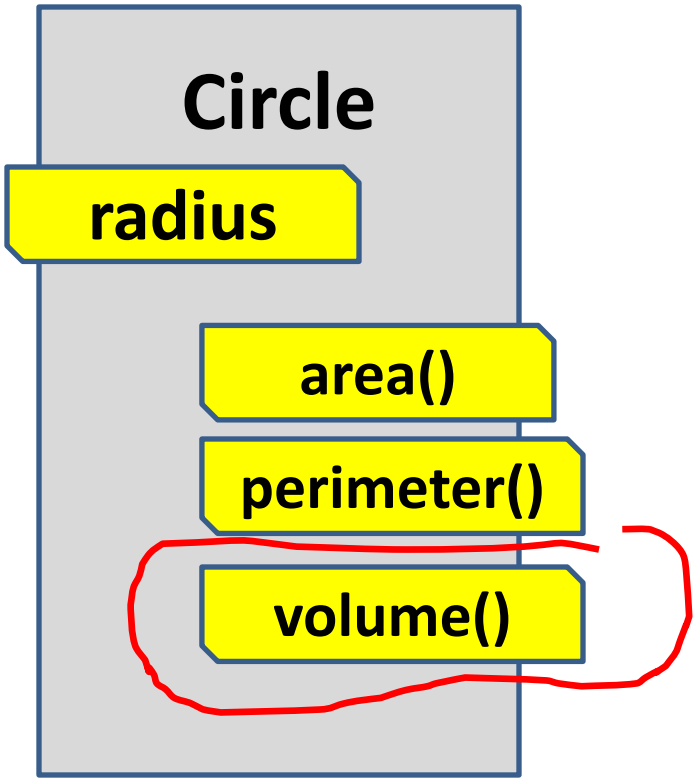
Low Coupling → better

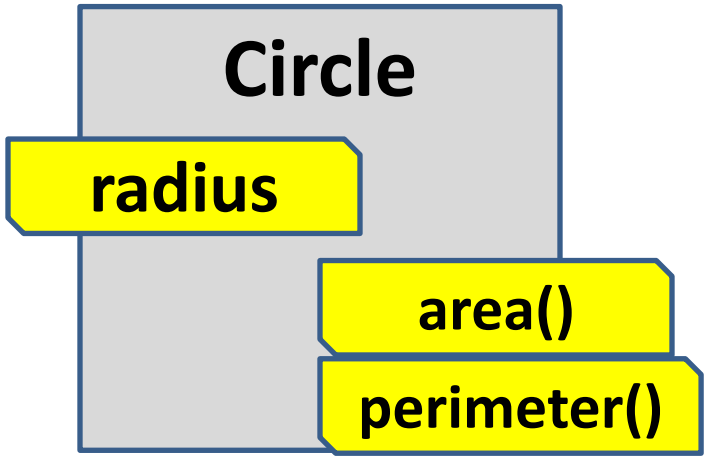
Cohesion

Cohesion is a measure of

“how strongly related and focused are the responsibilities of a class”

responsibilities → methods





High cohesion → better

Recap

Inheritance — **Derived class inheriting properties of Parent class**

Polymorphism — **An object behaving differently in different situations**

Encapsulation — **Putting the Data and methods together in a class**

Abstraction — **Using the data only via methods of the class**

Information Hiding

Recap

- Message passing — **An object Calling its method.**
- Coupling — **Measure of how much is 1 class dependent on the Other.**
- Cohesion — **Measure of How strongly related and focused are responsibilities of a class.**

Very Brief History : Java


James Gosling, the creator of Java

- **James Gosling** , Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.
- Initially called ***Oak*** , *then renamed as* ***Green*** , *finally as named as* ***Java***
- Sun Microsystems released the first public implementation as Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA).
- Oracle Corporation acquired “Java” from Sun Microsystems in 2009-10
- The Oracle implementation is packaged into two different distributions:

The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end-users, and the

Java Development Kit (JDK), which is intended for software developers and includes development tools such as Java compiler, Javadoc, Jar, and a debugger.

Remember ?

1995 , James Gosling from Sun Microsystems

The goal was to provide a simpler and platform-independent alternative to C++ , for Programming devices.

Is packaged into 2 different distributions :

1> JRE (Java run time Environment) → **Contains packages**

2> JDK (Java Development kit) → **Contains Javac**

Java comes 3 flavors :

1. J2SE (Java 2 Standard Edition)
2. J2EE (Java for Enterprise Edition)
3. J2ME (Java 2 Micro Edition)

Why Learn Java Basics?

Write less code: compared to C++.

Write better code: As Java is OOP's

Develop programs more quickly : As Java has many Build-in packages / classes

Avoid platform dependencies : As Java uses JVM

Write once, run anywhere : Due to the .class files

Distribute software more easily

Most important , to develop , Android Apps

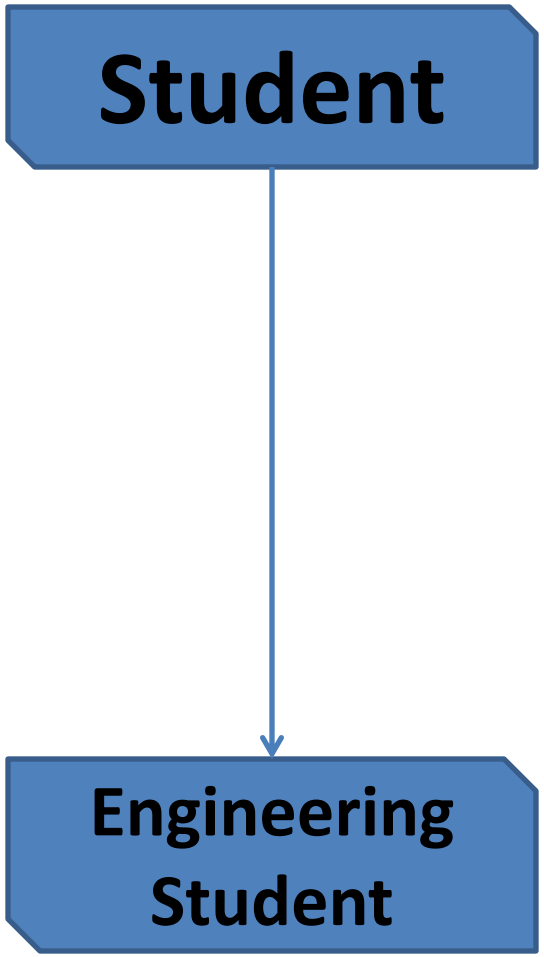
Important Concepts

Inheritance

4 - Types of Inheritance

1. Single or simple
2. Multi level
3. Hierarchical
4. Multiple

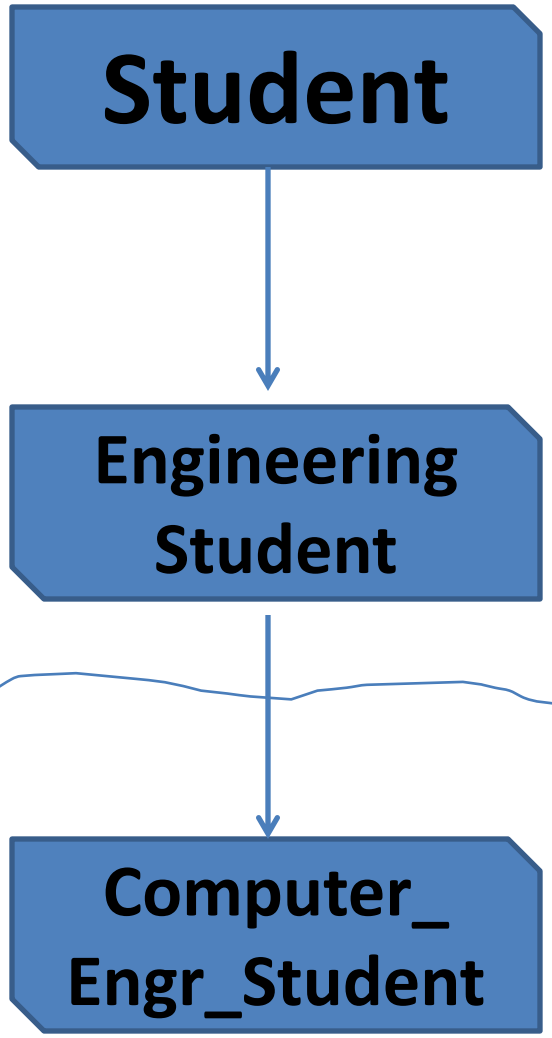
1. Single or simple




```
class Student {
  int roll_no;
  String name;
  int get(int p, String q){
  roll_no=p; name=q;
  return(0);
  }
  void Show(){
  System.out.println(name);
  System.out.println(roll_no);
  } }
```

```
class EngStudent extends Student {
  String spec;
  int get(String spec){
  this.spec = spec;
  return(0);
  }
  void Show(){
  super.Show();
  System.out.println(spec);
  } }
```

2. Multi level




```
class Computer_Engr_Student extends Eng_Student
 {
  String university;
  double marks_in_java_programming;
  int get(String u , double m){
  university = u;  marks_in_java_programming=m;
  return(0);
  }
  void Show(){
  super.Show();
  System.out.println(university);
  System.out.println(marks_in_java_programming);
  } }
```

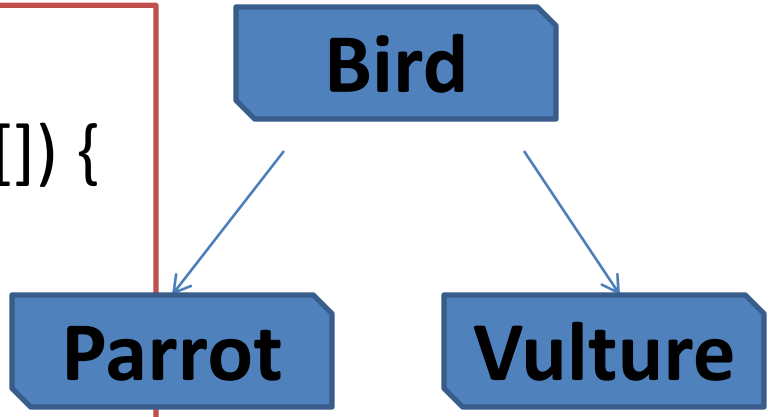
3. Hierarchical → 1 parent and 2 or more child

```
class Bird {
public void fly() {
System.out.println("Generally, bird fly"); }
}

class Parrot extends Bird {
public void eat() { System.out.println("Parrot eats fruits and seeds"); }
}

class Vulture extends Bird {
public void vision() { System.out.println("Vulture can see from high altitudes"); }
}
```

```
public class FlyingCreatures {
public static void main(String args[]) {
Parrot p1 = new Parrot();
p1.eat();
p1.fly();
v1 = new Vulture();
v1.vision();
v1.fly();
} }
```



4. Multiple : A class Inheriting from 1 base class and 1 or more Interfaces

Rule : we cannot use extends k/w twice

```
Class A  extends parent1  extends parent2  {
…
}
```

Not allowed

Soln : use Interfaces

```
Class A  extends parent1  implements interface1, interface2  {
……
}
```

Interfaces

Defn : Interface has abstract methods and constants.
i.e. it specifies what a class must do, but not how it does it.

access **interface** *name* {

access → public / default

return-type method-name1(*parameter-list*);
return-type method-name2(*parameter-list*);

type ***final*** *varname1 = value;*

final → constant

type varname2 = value;

....

final → by default

}

/* Area Of Rectangle and Triangle using Interface * /

```
interface Area  {
float compute(float x, float y);
}
```

Interface

```
class Rectangle implements Area {
public float compute(float x, float y)
{
return(x * y);
}
}
```

Class 1

```
class Triangle implements Area  {

public float compute(float x, float y)
{
return(x * y/2);
}
}
```

Class 2

```
class InterfaceArea
{
public static void main(String args[])
{
Rectangle rect = new Rectangle();

Triangle tri = new Triangle();

Area area;

area = rect;

System.out.println("Area Of Rectangle = "+ area.compute(1,2));

area = tri;

System.out.println("Area Of Triangle = "+ area.compute(10,2));
}
}
```

Object of interface

Referring to object implementing the Interface

Referring to object implementing the Interface

What is Abstract Class ?

For Self reading

1. Any class that contains 1 or more abstract methods must also be declared abstract.

2. There can be no objects of an abstract class. i.e., cannot use "**new**" operator.

3. 1 cannot declare abstract constructors, or abstract static methods.

4. Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared **abstract**.

4 Access Modifiers

private — Visible to the class only

Default or No Modifier — Visible to the package.

protected — Visible to all subclasses inside and outside the package

public — Visible to the world

Simple Example of Access Modifiers :

```
class Test {
int a; // default access
public int b; // public access
private int c; // private access
// methods to access c
void setc(int i) { // set c's value
c = i;
}
int getc() { // get c's value
return c;
}
}
```

Cannot be accessed outside the class. Hence methods needed.

Rule 1: A class can have attributes of all 4 access modifiers

Rule 2: for private → always code setters and getters

Using above class Test :

```
class AccessTest {
public static void main(String args[]) {
Test ob = new Test();
ob.a = 10;
ob.b = 20;

// ob.c = 100; // Error!

ob.setc(100); // OK
System.out.println("a, b, and c: " + ob.a + " " + ob.b + " " + ob.getc());
}
}
```

Allowed

Why ?

Rules for Identifier Declaration

Identifiers → used for

- ✓ class names
- ✓ method names
- ✓ variable names.

Java is case-sensitive, so **VALUE and Value are different**.

Some examples of valid identifiers are:

AvgTemp count a4 $test this_is_ok

Invalid variable names include:

2count high-temp Not/ok

- ✓ No special chars
- ✓ Don't start with digit
- ✓ No k/w allowed

Java Is a Strongly Typed Language

1st declare then use

Type conversions

Implicit or Automatic

When we go from lower to higher data type

```
int a;
byte b = 3;
a = b; // Implicit cast happens
```

Explicit

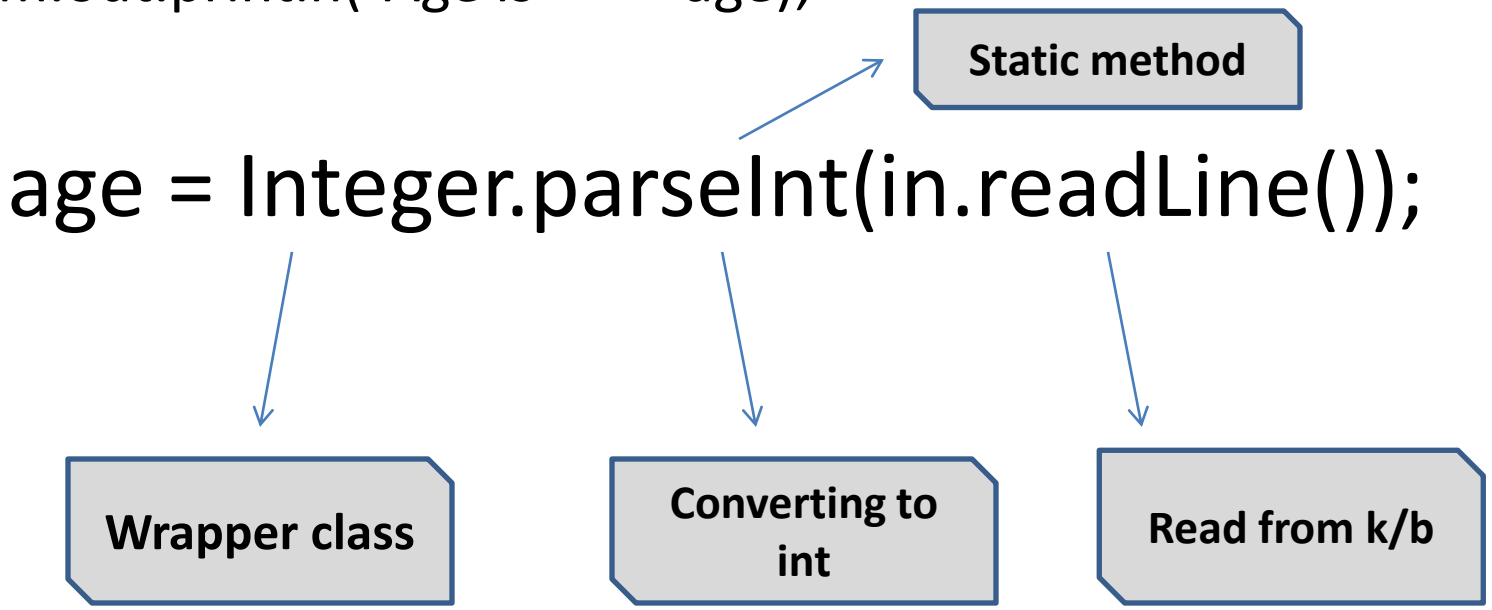
When we go from higher to lower data type

```
int a = 5;
byte b;
b = (byte)a;  // there could be loss of information
```

Wrapper Class

We use wrapper classes to convert from String to approp data type

```
System.out.println("Enter Age  ");
int age;
DataInputStream in = new DataInputStream(System.in);
age = Integer.parseInt(in.readLine());
System.out.println("Age is = " + age);
```



Rule: parse → convert

Most common parse methods

1. static int parseInt(String s)
2. static float parseFloat(String s)
3. static double parseDouble(String s)
4. static byte parseByte(String s)
5. static Long parseLong(String s)

---

parseX() → converts from String to Respective type X

Wrapper Class for each primitive type

Primitive data type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

TYPES OF ERRORS

Compile-time errors

All syntax errors.
These are detected by java compiler

```
Class Error1
{    public static void main(String args [])
        {
          System. out. println("Hello Java!")
} }
```

Run-time errors → **Exception**

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an incompatible class or type
- Attempting to use a negative size for an array

On an Exception →
java generates an error message and aborts the program.

```
Class Error2
{
        public static void main (String args [])
            {
                int a = 10;
                int b = 5;
                int c = 5;
                int x = a/(b-c);
                System. out. println ("x = " + x);
                int y = a/ (b + c);
                System. out. println ("y = " + y);
            }
}
```

Exception : divide by 0

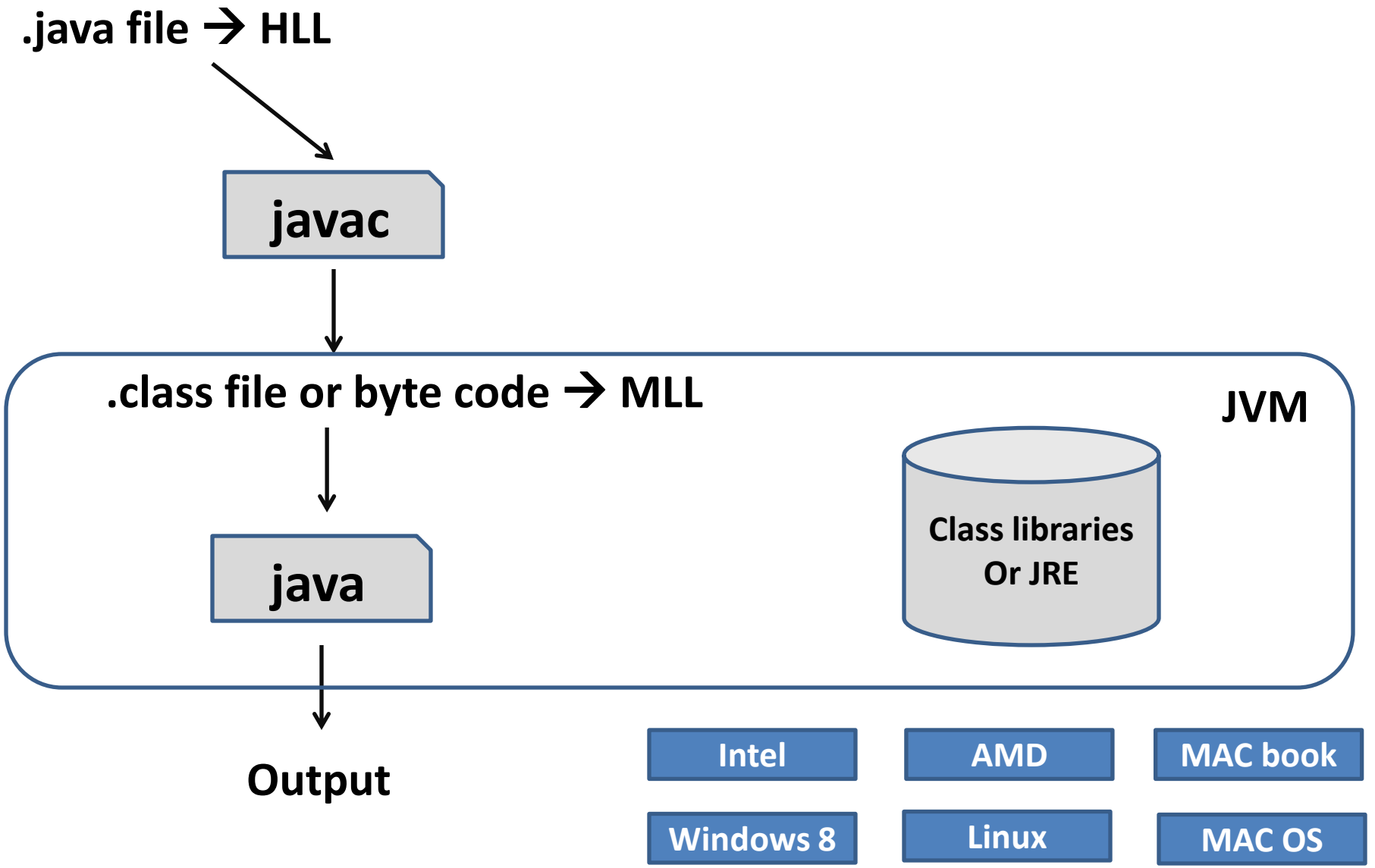
Simple 4 steps to handle exception

1. **HIT** the exception
2. **THROW** the exception
3. **CATCH** the exception
4. **HANDLE** the exception

Use try-catch block

```
Try                                                    → Hit and throw
{
        Statement;      // generates an exception
}
Catch (Exception-type e)                               → Catch And Handle
{
        Statement;      //processes the exception
}
```

What is JVM ?


.java file → HLL
javac
.class file or byte code → MLL
JVM
java
Class libraries
Or JRE
Output
Intel
AMD
MAC book
Windows 8
Linux
MAC OS

Many Advantages of JVM or 2 step Process

- Secure — **Src code is not given, .class file is released**
- Portable — **Compile on 1 pc and run on another**
- Architecture-neutral — **Runs on all OS**
- Interpreted — **2nd step called Interpretation**
- High performance
- Distributed — **.class files can be mailed and run on remote pc**

Lets start with Android OS Programming