Core Java:

Modifiers:

1. Access modifier: public( to all), protected( within the same package (any ref)and child classes in a diff package with child reference. Not with the parent reference), private(this class), default (package level: any ref)

package java1;

class A{

protected int x=4;

int y= 5;

}

Package java2;

Class B extends A{

void print(){

A a= new A();

//a.x;

this.x;

B obj= new B();

Obj.x;

}

}

Access modifier: class: default/ public. Methods/ instance variables/ static. Not with local variables.

Variables:

1. Local: within a block/method
2. Instance: belongs to object of the class
3. Static: belongs to the class, that means the copy of this variable is shared among the objects. We should refer to them by the class-name.

Class A{

int y=5; //object of the class

static int abc;

{ //instance block

int x=5;  
}

void print(){

Sysout(x);

Int z=6;

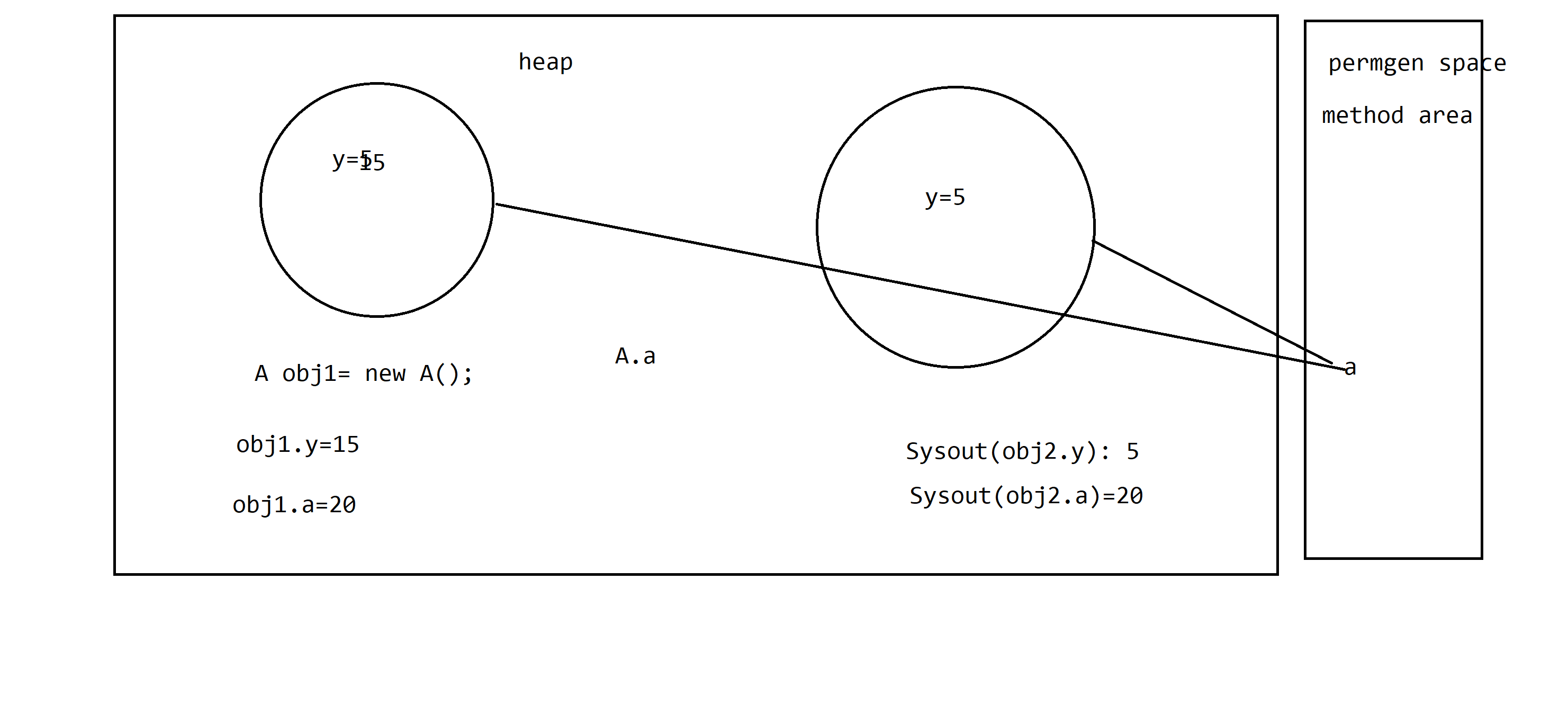
}

Void m1(){

Sysout(z);

}

}



Modifiers:

1. Static modifier: class/method/variable/block

Static variable: belongs to a class, copy is shared among all objects

A obj1= new A();

A obj2= new A();

Obj1.x=20;

* A.x = 20;

Class A{

Static int x;}

Static method: Belongs to a class, can access it directly by the classname.

Class A{

int x=4;

void print(){//depends on the state of the object, I cannot make it static

Sysout(x);

}

}

A obj1= new A();

Obj1.print(); //4

A obj2= new A();

Obj2.x=10;

Obj2.print(); //10

In a static block/method, u cannot access non-static members.

Instance block: It is called just before the constructor.

Static block: It is called only once when the class is loaded into the JVM.

Class A{

A(){

Sysout(2);

}

A(int a){

Sysout(3);

}

{  
Sysout(1);

}

}

A a= new A(); // 1 2

A b= new A(3); // 1 3

Class A{

Static {

Sysout(1);

}

{

Sysout(2);

}

A(){

Sysout(3);

}

}

A a= new A(); //1 2 3

A b= new A(); // 2 3

**Loading a class: main**

Class A{

Public static void main(String ars[]){

B b;

}

}

Class B{

Static void print(){}

}

//B would not be loaded

Main(){

B a= new B();

B.print();

}

**Final keyword:**

1. Class
2. Method
3. Variable

Bydefault, instance and static variables are initialized to their default values

Boolean: false

Int:0

Float: 0.0f

String: null

String str;

Local variables should be initialized before they are used.

Class A{

Int y; //0

void print(){  
int x;

Sysout(y);

}  
}

Final instance variable should be initialized before an object is created

//declaring/ instance block/constructor

Class A{

final Int z;//=5;

A(){

//Z=5;

}

void print(){

Z=5;//error

}

}

Final static variables: It should be initialized before the class is loaded

Static block/ declaring it.

Class A{

final static int z;

{

Z=10;//error

}

}

Static, final

OOPS:

Classes, relationships between them

1. Is-a: Inheritance
2. Has -a: Association

Ravi has a car

Ram is a manager of Shyam

Class Person{

String name, address, phoneNo;

Car car;

Talk(), study(), eat();

}

Class Car{

Wheels, brakes, gears;

}

Class Employee{

Int id;

String name;

Float salary;

Car car// has a : association

}

Class Manager extends Employee{ //Is-a, implements

String project

}

//OOPS: real world entities: object, relationship

Ram is driving santro/wagonr

Person: object: Ram

Car: object: Santro/WagonR

Banking app: Checking/ Saving/ Customers/ Manager/Loan

Account: Checking/ Saving

Procedural: C language

**Pillars of OOPs:**

1. Abstraction: abstract: interface
2. Inheritance: extends/ implements
   1. Class extends Class1
   2. Interface extends Interface1, interface2
   3. Class implements Interface1, Interface2
3. Polymorphism: method overriding/ overloading
4. Encapsulation: access modifiers
5. **Abstraction:**

Abstract: method/ class

Abstract method: method which has no body.

Abstract void print();

Abstract class:

We cannot create an object of an abstract class.

abstract class A{

}

Can I declare class to be abstract even if it has no abstract method? Yes

Banking application:

Abstract Class Account{}

Class Saving extends Account{}

Class Checking extends Account{}

Interface: These are 100% abstract classes.

Interface I1{

float PI=3.14f; // final, public, static: constant

void print(); //by-default abstract public

}

Java 8:

Interfaces: default, static methods

Java9:

Private methods

Interface Shape{

//static method

}

Abstract class{}

Class Square{

}

Class Rectangle{

}

//static factory method

Interface Printable{

Void print();

default void edit2(){

//blank body, backward compatibility

m1();

-----

}

default void edit1(){

m1();

-----

//blank body, backward compatibility

}

// y private methods in interface?

private void m1(){

Sysout(“in default”);

}

}

Class PrintToConsole Implements Printable{

//print

}

Class PrintToFile implements Printable{

//print

}

//new functionality : editable, void edit();

**Inheritance:**

Extends/ implements (Is-a)

**Polymorphism: The name of methods shud be same**

1. Method overloading:
   1. The arguments should be diff (number, order, type)
   2. Return type, exceptions, access modifier does not matter
2. Method overriding:
   1. The arguments should be exactly same
   2. Return type shud be same/ covariant(parent child)
   3. Access modifiers should be same/ less restrictive
   4. U cannot throw a new checked exception

Class A{

void add( int a, int b){}

void add( int a, float b){}

}

Public> protected>default>private

Class B extends A{ //3 add method

void add( float a, float b){} //overloading

private void add( int a, int b){}//error

//int add( int a, int b){}//neither overloading/overriding: error

}

**Encapsulation:**

Properties: private: getters/ setters

Class A{

private Int x;

public setX(int x){

//validations

this.X=x;

}

public int getX(){  
return x;

}

}

A obj= new A();

Obj.setX(10);

Obj.setX(5); //obj.x=5;

<https://github.com/payalbnsl/Corejava-reference>

<https://github.com/payalbnsl/capgemini-material>

M2.zip: core java

**Singleton**: U can create only 1 object of this class.

**Immutable**: Once u create the object of the class, u cannot change the state of it.

1. Singleton:

class A{

}

A obj1= new A();

A obj2= new A();

1. Class A{

private static A obj= new A(); //Eager initialization, object created when class is loaded

private A(){

}

private A getInstance(){

return obj;

}

}

1. Class A implements Cloneable{

private static A obj; //Eager initialization, object created when class is loaded

private A(){

}

private static A getInstance(){

if(obj== null){

obj= new A();

}

return obj;

}

Public A clone(){  
return obj;

}

}

A obj= A.getInstance();

Obj.clone();

1. Clone
2. Deserialize
3. Multithreaded

**Cloning:**

1. Deep cloning
2. Shallow cloning

Object: protected Object clone(){

//shallow copy of the object

}

Class A extends Object implements Cloneable{

Int x=7;

String a=”hello”;

List<String> list= new ArrayList<>();

public Object clone(){

//super.clone();//deep copy

A obj1= new A();

Obj1.x= x;

Obj1.a = a;

Obj1.list= new ArrayList();

}

}

Class B{

A obj= new A();

A obj1=obj.clone();//error : CloneNotSupportedException

Obj1.list.add(“hi”);

}

Marker interface: interface without any method

Cloneable: contract, if u clone an object, the class should implement Cloneable.else it will throw CloneNotSupportedException

**Serialization:**

Java object: save the state: file? Transmitting over the network.

Convert into bytes: serialization

Byte stream to java: deserialization

1. Class shud implements Serializable/Externalizable.: Not Serializable exception

ObjectInputStream/ ObjectOutputStream : readObject , writeobject

Externalizable: readExternal, writeExternal

Serializable: marker

**IO: io package**

Reader/ writer (char: 2/4 bytes: text data): i/o streams(1 byte: 8 bits: binary: video/ audio/ image)

FileReader, FileWriter, FileInputStream, FileOutputStream

BufferedReader: buffer: store more than 1 char

PrintWriter: primitive data: float/ binary

Scanner

ObjectInputStream/ ObjectOutputStream: objects