Date

Module - 1 : Object Orientation.

1) Structure Of a class.

1) Class has instance variables, methods and local variable.

```
class A
{ int x; // instance variable.

  void setX (int x) // Local variable.
  {
    this.x = x;

    x = this.x;

    this.x = this.x;
  }

}
```

4
7

Rules.

1) Instance variable by default are initialize to zero for int and null for string, 0.0 for float, 0.0 for double, 0 for long.

2) Local variables declared inside method have scope only within that method.

3) Local variables are not initialized by default, by JVM; if it declare a local variable without initializing it gives us a warning. But if we use that variable without assigning a value we get compilation error.

4) To differentiate b/w local and instance variable having same name we use this pointer.

Date

2: Heaps and Stacks (Memory)

// Program

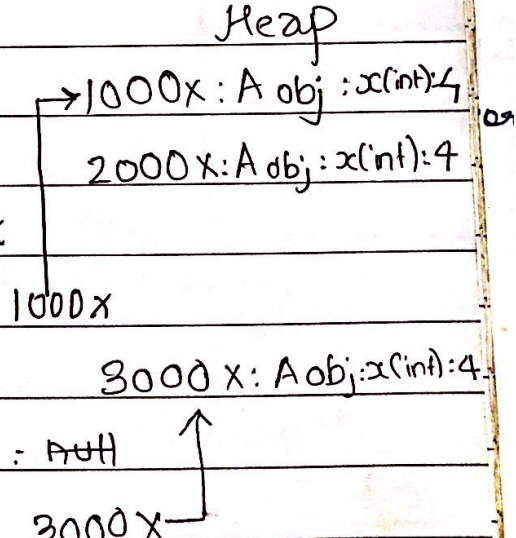
```
class A
{
  int x = 4;

  void setX(),
  {
    x = 5;
  }

}

class App
{
  psvm(--)
  {
    A a = new A();
    new A();

    A a1;
    a1 = new A();
    Sop(a.x); // 4
```



#_____
#_____

Date

Heap.

a = a1;
3000x

a1.x = 7;

sop(a.x); // 7

A a3 = a1;

a1 = null;

sop(a1.x);

Run. Excep →

1000x: A obj: x(int): 4

Stack 2000x: A obj: —u—

3000x: —u— :x(int): ~~4~~ 7

A a: ~~1000x~~ ~~3000x~~ 3000x

A a1: ~~null~~ ~~3000x~~ null

A a3: 3000x

Date

3: Overriding & Overloading.

Rules.

The two methods are considered to be overridden if fou. rules apply.

i) The names of the methods should be same.

ii) No. of arguments in the methods should be same.

iii) Data type of arguments should be same.

iv) Sequence of arguments should be same.

Rules:

i) If there is a valid overriding.

— The return type of both method should be same.

— We cannot ~~cannot~~ come from higher to weaker access privilodge.

What makes you happy?
#HappyCollegeDays

#_____
#_____

Date

Access Privilege Hierarchy.

1) Public

2) Protected

3) Default

4) Private.

Over Loading

→ If the names of mth are same and there is no valid overriding it is considered to be overloading.

Date

// Program Ex:

```
class A {
{
   int x = 4;
   void m1 (int x)
   {
     Sys. out pln ("A");
   }
}

class B extends A
{
→ private void m1 (int y) ==
   {
     Sys. out. pln ("B");
   }
}
```

Error

{ Default → Private (weak access specifier) }

What makes you happy?
#HappyCollegeDays

#_____
#_____

Date

Encapsulation

Eg:

```
Class Customer
{
   private int balance; // marked as
                        // private as per
                        // the suggestion.

void deposit (int amount)
  {
   this.balance = this.balance + amount;
  }

void withdraw (int amount)
  { this.balance = this.balance - amount;
  }

public int getBalance ()  // getter
  {
   return balance;
  }

~~public~~ voi
```

What makes you happy?
#HappyCollegeDays

#

Date

```
//setter.
public void setBalance(int balance)
{
    this.balance = balance;
}
```

Rules for Encapsulation

1) All instance variable should be marked as private.
2) We should provide getters and setters for all variable.

Polymorphism:

It means multiple forms of an object.

A
↓
B
↓
C
↓
D

```
A a = new B();   ✓
A a1 = new C();  ✓
B b1 = new D();  ✓
C c = new B();   ✗
C c = new C();   ✓
```

Rules for Polymorphism:

1) If a variable is called Polymorphically, we follow the reference.

Date

~~Example;~~

2.) If a method is called polymorphically we do following things,

i) we go to superclass and check ~~Weather~~ ~~ff~~ the mth signature is present or not, if ~~so~~ it is present then we come to point (ii)

ii) we come to subclass and check wether the mth is overridden, if it is overridden we execute that method.

If it is not overridden we execute the mth written in superclass.

#

#

Example;

```
class A
{
  int x = 5;
  void display()
    {
      sop (x);
    }
}

class B extends A
  {
    int x = 6;
    void display()
    {
      sop (x);
    }

psvm ()
{
  A a = new B();
  sop (a.x);    // 5.

  a.display(); // 6
}
}
```

What makes you happy?
#HappyCollegeDays

Example (2):-

```
class A
{
  int x=5
  void display()
  {
    sop (x);
  }

  A getObj ()
  { return new B();
  }
}
class B extends A
{
  int x=6;
  void display()
  { sop (x);
  }

  psvm()
  {
    A a = new B();
    sop (a.getObj().x);   // 5
    a.getObj().display();  // 6
  }
}
```


#_____

#_____

Date

Statics

1.> static class
2.> —u— var
3.> —u— mtd
4.> —u— block
5.> Anonymous block

1.> Static Class.

A class cannot be marked as static unless its an inner class.

Ex: static class A { } X

invalid

2.> Static Variable.

If a var is marked as static it is stored in special memory called as statics.

This means we can reach out to this variable in 3 ways

#

Date

1.) Direct (x)
2.) Using class (A.x)
3.) Using Object (a.x)

And hence static contents are said to belong to class as they are stored in statics memory on the class and not on global heap.

ii) static variables as like instance variable are by default to zero and can also be initialized or reassign a value inside a static or an anonymous block.

#

Example:

```
class A{
{
  static int x = 5;

  psvm()
  {
    A a1 = new A();
    A a2 = new A();
    a1.x = 6
      x = 7
    sout(a2.x); //7
  }
}
```

Stack

A a1 : 1000x

A a2 : 2000x

Heap

1000x : AObj : 100x

2000x : AObj : 100x

A statics.

100x : x(int) : 5/6/7

4.) Static Block.

i) Static blocks are used to initialize static variables.

ii) Static block is executed when the class is loaded in the memory. That means when we write psvm.

Example.

```
class A
{
static int x = 5;
static {
        x = 6;
        sout (x);
       }

psvm ( ),
  {

  }
}
```

Output : 6

What makes you happy?
#HappyCollegeDays

5.) Anon Block.

i) Anonymous are called when class is instantiated that means when we create an object.

ii) These blocks are used to initialize non static instance variables as well as static variables.

Example.

```
class A
{
  Static int x;
  int x;
  Static {
          x = 6
          Sout (x);
        }
  {
     y = 7;
     Sout (y);
   }
psvm () { new A ();}
}
```

O/P : 6
7

What makes you happy?
#HappyCollegeDays

#_____
#_____

Date

3.> Static Method.

i) A method is marked as static if we do not want this to be overridden.
However at ~~the~~ if the subclass wants to use the same signature of mtd it can do so by marking mtd as static.

ii) This means that both classes have their own individual copies of mtd as they are both marked as static.

Example:

```
class A
{ int x = 5;
   static void display()
    {
      Sout("A");
    }
}
```

What makes you happy?
#HappyCollegeDays

#_____
#_____

Date

```
class B extends A
{
  int x=6;

  static void display()
  {
    sout("B");
  }

  psvm()
  {
    A a = new B();
    a.display();
  }
}
```

Date

Note:

1) Non-static variable cannot be accessed from static context, but static variable can be accessed into non-static context

2) We can have multiple static blocks and anonymous blocks. Execution ~~we can have multiple~~ follows the sequence of blocks.

3) If we create 5 obj of class A, the anonymous block gets called each time the obj is ~~dead~~ created.

Date

final

1.> final class
2.> final var
3.> final mtd.

1.> final Class.

If a class is marked as final it cannot be inherited.

Example:

```
final class A
{ }
class B extends A   X
{ }
```

(cannot be inherited)

2.> final variables.

1) If a variable is marked as final it has to be initialized
Since by default jvm does not initialize it.

Date

Example:

final int x = 5; ✓

final int y ; X

(iii) A variable can be initialized in three ways.

(i) Explicitly.
(final int x = 5;)

(ii) Inside the blocks.
- Static
- Anonymous.

(iii) Inside a constructor

Example: (1)

```
class A {
  final int x;
  {
    x = 5; // initialized.
  }
A () {
      x = 5; // assignment not
               allowed
    }

psvm ()
{
  new A();
}

}
```

Example ②:

```
class A
{
    final static int x;
    {
        x = 5; // cannot initialized.
    }

    static {
        x = 5; // initialized
    }

    psvm ()
    {
        new A();
    }

}
```

Anon Block

3> final Mtd.

If mtd is marked as final, the signature of the method gets locked, which means that no other subclass can use the same signature.

Note:

1) final methods can be inherited.

Example:

```
class A {
    final void display()
    {
    }
}
class B extends A {
void display() { }   X
final void display() { }   X
static void display() { }   X
```

→ display signature is locked with A.

```
void display(int x) { }
```

↳ signature is different

- Abstract Class & Abstract Mtd

1) Abstract Class

i) A class is marked as abstract, so that we can write abstract method inside it.

ii) A class marked as abstract, cannot be instiansiated.

Eg:

```
abstract class A {
    psvm (—) {
        A a = new A( );   ✗
    }
}
```

What makes you happy?

#

#

Date

iii) A abstract class can have abstract as well as non - abstract variables.

Eg:

```
abstract class A {

    abstract void display(); // abst - mtd

    void m1() { } // Non - abstract mtd

}
```

iv) If a class is marked as abstract and has abstract mtd then it is necessary for subclass which extends its abstract class, to override all the abstract ~~class~~ mtd of superclass.

If the subclass does not wish to override the abstract mtd, it can also marked itself as abstract.

Date

Abstract Methods.

i) Abstract method does not have a body.

Eg: abstract void m1();

ii) Abstract mtd can only be written inside an abstract class.

Example:

```
package p1;

abstract class A {
    abstract void m1();
    void m2() {
        sout("M2");
    }
}
abstract class B extends A
{
    ~~ab~~ void m3()
    {
        sout("m3");
    }
}
class C extends B {

    void m1() {

    }

}
```

Date

Case Study:

There are 3 programmers working on same package.

- A wants to create 2 mtd's m1() and m2() but knows the implementation of only m2(). ↓ all using reference
- A also wants m2() to not be overridden.
- It wants to create a variable (x) & gives a constant value 5.
- B wants to implement method m3() & knows the code as well. It also wants to implement m4() but does not know the code.
- C has no mtd's of itself, but is not happy with m2() & wants to rewrite it.
- Create main() in class C & call all the ~~not~~ mtd's using C object.
- Call (x) directly.

#

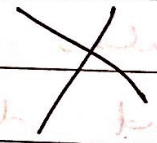
Date

Interface:

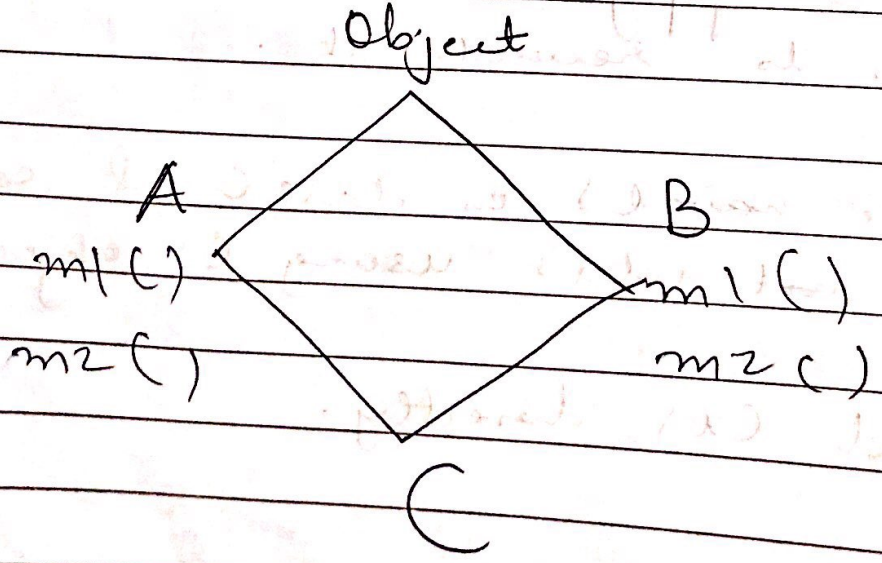
Need for Interface in Java.

Ex 1

```
class A {
    m1() { }
    m2() { }
}
class B {
    m1() { }
    m2() { }
}
class C extends A, B
```



Death of diamond



Date

Ex 2

class A { }

class B { }

class C extends A implements B { }

Rules

i) Interface is an abstract entity and can also be declared as ~~inter~~
(abstract interface A)

ii) A mtd inside interface by default is public and abstract.

Eg: interface B {
void m1();
}

→ public abstract void m1();

What makes you happy?
#HappyCollegeDays

#

#

Date

iii) A variable in the interface is final and static by default.

Eg: int x = 5; is written as

static final int x = 5;

iv) As the interface is an abstract entity it cannot be instantiated.

Eg: B b = new B(); ✗

(v) If a class implements an interface it has to compulsorily override all the abstract mtd of interface.

If the class does not wish to override all the mtd, it can mark itself as abstract.

Date

Example:

```
package p1;

abstract interface B {

    int x = 8; // public final static int x=8;

    void m1(); // public abstract void m1();

}

abstract class C implements B {

    public abstract void m1();

    abstract void m2();

}

class D extends C {

    public void m1() { }
    void m2() { }
    static B b;

    psvm() {
        sout(x);
    }
}
```

What makes you happy?
#HappyCollegeDays

#
#

Hidden Rules:

i) A var inside interface are static we can access them in three ways.

① Directly.

② Using an interface name (B.x)

③ Using an reference.

Ex sop (b.x);

If the reference is created outside of main() we have to mark it as static, to access in static context.

If the reference is created inside main(), it becomes local reference, so has to be initialize to null.

(ii) A interface can extend the other interface.

Ex:

```
interface A {

    void m1();

}

interface B extends A {

    void m2();

}

class C implements B {

    psvm(-) {

    // implement (override) m1() & m2().

    }

}
```

Date

Death of Diamond Scene

BB

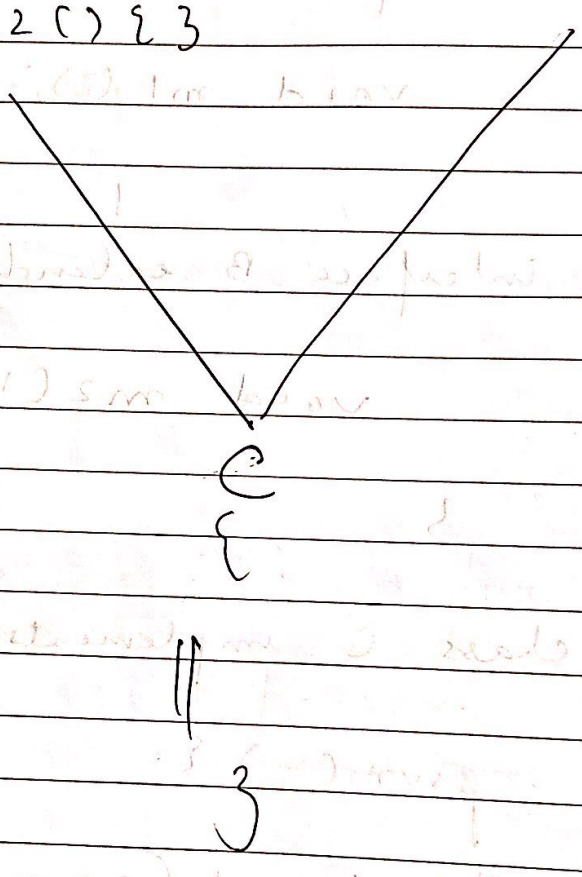
void m2();

B

A

void m1(){}
void m2(){}

void m1();



C
{
||
}

Program ↳

#
#

Date

```
package p1;

interface BB
  {
    void m2();
  }
interface B extends BB
  {
    void m1();
  }
class A {
   public void m1()
    {
      Sout("A");
    }
   public void m2()
    {
      Sout("B");
    }
  }
```

} copy

```
class C extends A implements B {
  psvm(-) {
          C c = new C();
          c.m1();
          c.m2();
        }
}
```

What makes you happy?
#HappyCollegeDays

#_____
#_____

Date

(iii) Interface can also be used polymorphically

{

} paste.

```
class C extends A implements B
{
  psvm(-)
  {
    B C = new C(); // polymorphic object
    C.m2(); // m2() is accessible from B.
  }
}
```

Constructor

1) Constructors are called by JVM when we instantiate a class.

2) A class can have one or many constructors.

3) If a programmer does not write the constructor inside the class, by default JVM creates no argument default constructor for us.

Ex:

Programmer

```
class A
{
void display(){}
}
```

JVM

```
class A
{
  A()  // Default
{       constructor
super(); // calls constructor
}          of super class.
void display(){}
}
```

What makes you happy?
#HappyCollegeDays

#
#

Date

4) A constructor can be marked as public, protected, default, private.

However, marking a constructor private means we cannot instantiate our class in any other class, which negates the law of inheritance and hence, should never be done.

5) Every constructor has a statement called super(); as its first line of the code which calls the constructor of super class.

This super can be written by either JVM or programmer, So this means that if Programmer writes super(); JVM will not write it and if Programmer does not write super(); jvm will write default (super();)

#

Date

Constructor Chaining:

Consider the foll scenario.

```
class C extends B { }

class B extends A { }
```

Rule:

i) If we create an object of C, we have to make sure that atleast one constructor of all super classes gets called.

Note: A prog.. can write a logic for calling multiple constructors of one class using super() and this().

this()

this() mtd calls the constructor of the same class.

Eg: i) this(5);

calls the constructor of same class with one argument.

ii) super(3, 5);

calls the construction of super class with two arguments.

iii) super();

calls the construction of super class with zero arguments.

2) this() and super(), both should be first statement of the mtd, and hence they cannot be present in same constructor.

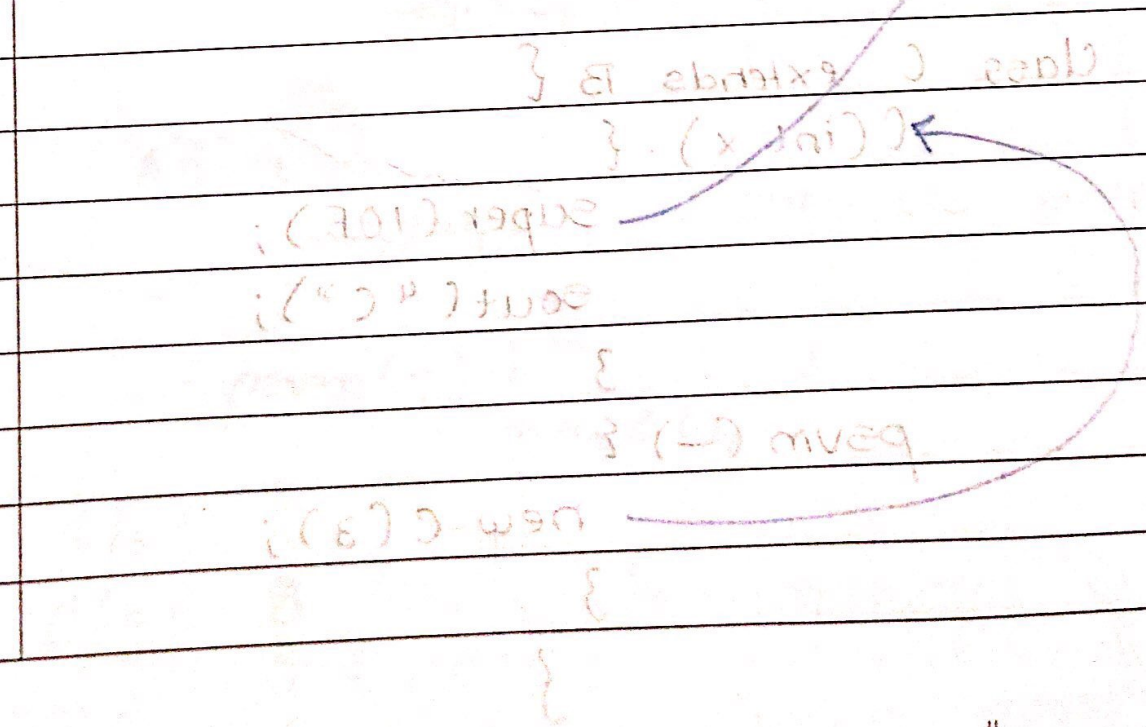
Date

Constructor Calling Rules.

1) A constructor can be called only in 2 ways.
   i) When class is instansiated.
   ii) From within the other constructor.

Note: A constructor cannot be called from other normal method, and hence we cannot write super() or this() inside a normal method.

However, we can call a normal method from a constructor



#

Date

Constructor Chaining Example:

```
class A {
    A(int x) { super();
               sout("A");
    }
    A() { this(5); }
}

class B extends A {
    void B() { }
    B() { super(6); }
    B(float y) {
        this();
        sout("B");
    }
}

class C extends B {
    C(int x) {
        super(10F);
        sout("C");
    }
    psvm(-) {
        new C(3);
    }
}
```

→ object().

Date

Constructor Calling Example : 2.

```
class A {
    void display () {
        sout ("A");
    }
    A () {
        // this() -- cannot call constructor recursively.
        display ();
    }
}
class B extends A {
    int x = 6;
    void display () {
        // this(); -- not allowed as mentioned above.
        sop ("B");
    }
    B () {
        display (); // overridden method.
    }
    psvm (-) {
        new B ();
    }
```

O/P
B
B

Note: If display marked as static O/P: A
B

What makes you happy?
#HappyCollegeDays

Date

Reference Variable Casting.

Rule.

1) Superclass Ref = 'sub class | Ref. ✓valid

- called as Implicit Cast.

2> Subclass Ref = superclass Ref ✗ invalid

Ex:

```
class A { }

class B extends A {
    psvm (-) {
        A a = new A();
        B b = new B();
        a = b;        ✓
        b = a;        ✗
    }
}
```

Date

Example: (1)

```
class A {

}

class B extends A {

    psum(-)
    {
        A a = new A();
        A a1 = a;
        B b = (B) a1;
        B b1 = new B();
        b = b1;
    }
}
```

Rules:

80)

Date

Ex: 1

```
class A
{
  int x = 5;
  void display()
  {
    sout (x);
  }
}
class B extends A
{
  int x = 6;
  void display()
  {
   sout (x);
  }
  psum(-)
  {
   A a = new A();
   B b = new B();
   a = b;
   sout (a.x);
  }
  a.display();
  A a1 = new A();
  B b1 = (B) a1;
  b1 = new B();
  b1.display();
 }
```

What makes you happy?
#HappyCollegeDays

Scanned with CamScanner

Date

Rules.

i) In casting if heap & stack get left in unstable state we get class cast.

ii) A ref var which is in an unstable state should not be directly marked as null. However it is valid to assign null value.

iii) Unstable state is recorded in the log and hence downcasting should be done only when it is required.

iv) Implicit cast converts pure object into a polymorphic object and so the rules of polymorphism apply.

Date

Garbage Collection

1) When an object is created the object entry goes in the heap, whereas refrences are stored on the stack.

2) The only way to reach out to heap is by going thru stack.

If there is an entry on heap, which we cannot reach out from stack, it is considered to be eligible for garbage collection.

Eg:

```
class A
{
int x = 10;
psum() {
A a = new A();
A a1 = a ;
a = null;
a1 = null;
// here → How many objects are garbage collection Ready?
}
}
```

Heap

1000x : A obj : 2000 x

2000x : x(int) : 10

Stack

A a : 1000x | null

A a1 : 1000x | null

Ans:- 2

Date

Example:

```
class Book
{
  Integer isbn = 10;

  psvm(-)
  {
    Book b1 = new Book();
    Book b2 = b1;
    b2.isbn = 15;
    Book b3 = change_isbn(b2);
    System.out.println(b3.isbn);
    // here. → How many eligible for
                garbage collection (4)
  }

  Static Book change_isbn(Book b)
  {
    b = new Book();
    b.isbn = 21;
    Book b1 = new Book();
    return b1;
  }
}
```

Date

Change_isbn ~~main sta~~

stack

Book b: 1000 x | 4000 x

Book b1: 7000 x

main-stack

Book b1: 1000 x

Book b2: 1000 x

Book b3: 7000 x

Heap

✓ 1000x: Bookobj : 3000 x

2000 : isbn(int) : 10

✓ 3006 x: isbn(int) : 15

4000 x : Bookobj : 6000 x

5000 x : isbn(int) : 10

6000 x : isbn(int) : 21

✓ 7000 x : Bookobj : 8000 x

✓ 8000 x : isbn(int) : 10

Date

Working Of Garbage Collection

```
class Person
{
   int id;

psvm(-)
{
Person p1 = new Person();
Person p2 = new Person();
p1 = null;
p2 = edit(p1);
Sout(p2.id);
p2 = null;
// here ??
}
static Person edit(Person p1)
{
   p1.id = 16;
Person p = new Person();
   p1 = p;
   return p1;
}

}
```

What makes you happy?

#