Encapsulation

OPP's concept

1. Encapsulation
2. Inheritance
3. Polymorphisms
4. Abstraction

Encapsulation:

- Primary is security
- Data binding and hiding

Inheritance:

- Code reusability

Polymorphism:

- Code flexibility

Abstraction:

- Implementation hiding, only features visible

Encapsulation:

- Data binding and hiding
- Providing security
- Providing controlled access to members
- Keywords :
- Private ,setters , getters , this keyword, shadowing

new object creation -- instantiation

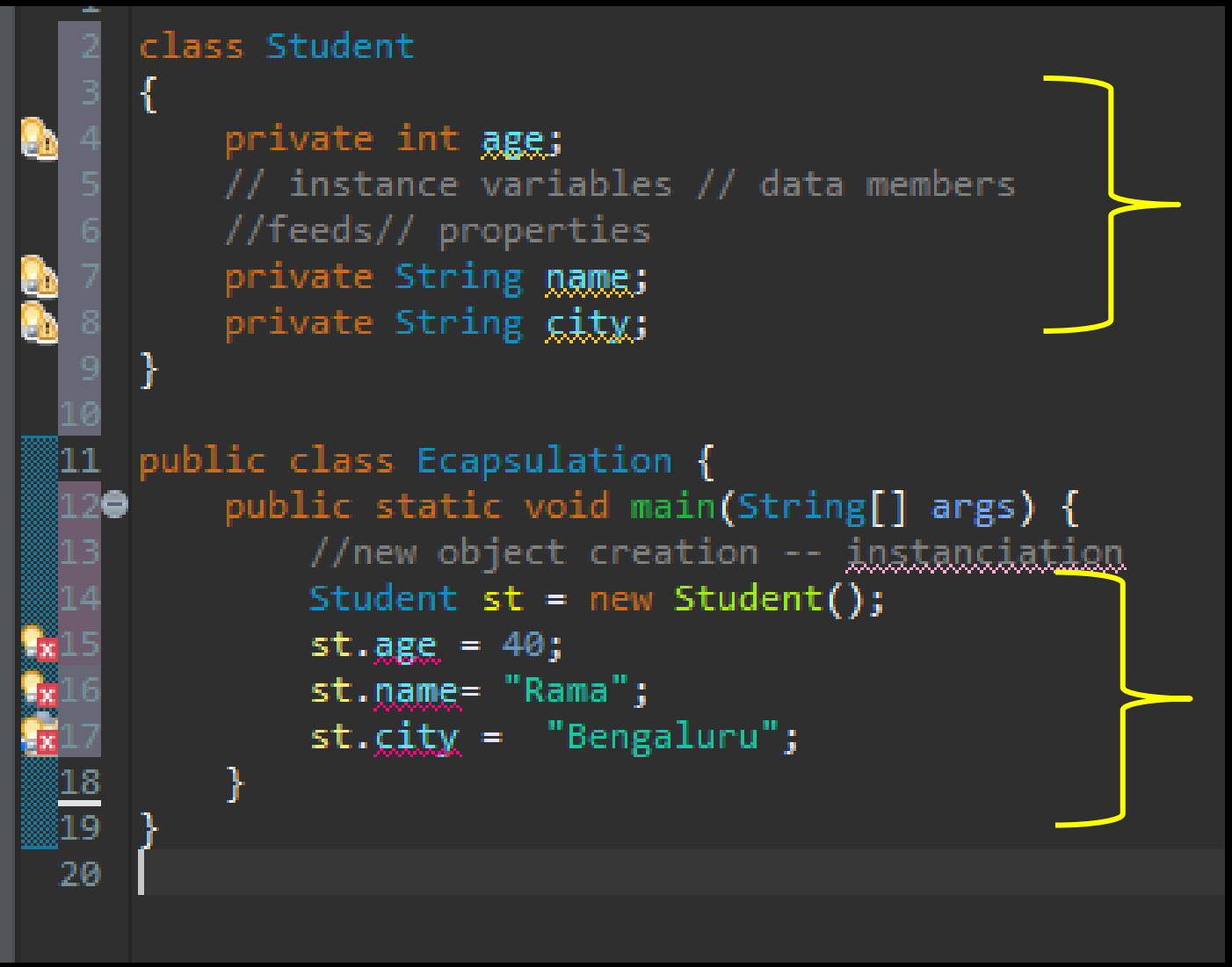
```java
class Student
{
    int age;
    // instance variables // data members
    //feeds// properties
    String name;
    String city;
}

public class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();
        st.age = 40;
        st.name= "Rama";
        st.city =  "Bengaluru";
    }
}
```

But how? →

Need to provide conditional access

Variables can be accessed directly → Not a good programming, no security

Private keyword



Due to private , variables can be accessed in this class only

variables can't be accessed in this class

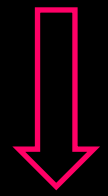
Now we require the data members to be accessed from outside, as they are created with some purpose

We need to provide values to the variables from outside.

Create a method inside the class , where the variables are available

So, method needs to be created

Activity



So, method needs to be created

```java
class Student
{
    private int age;
    // instance variables // data members
    //feeds// properties
    private String name;
    private String city;

    //method creation
     void setAge(int a)
    {
        age = a;
    }
    int getAge()
    {
        return age;
    }
}

public  class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();
        st.setAge(40);
        int age = st.getAge();

        System.out.println(age);
    }
}
```

Setters

Getters

For every variable you need to have setters and getters

Set age

Get age and transfer it to variable age

4 steps needed to get information using setters and getters:

1. Create setter method:

```
void setAge(int a)
{
    age = a;
}
```

2. Create getter method:

```
int getAge()
{
    return age;
}
```

3. Set the value

```
st.setAge(40);
```

4. Get the value and give it to variable

```
int age = st.getAge();
```

```java
class Student
{
    private int age;
    // instance variables // data members
    //feeds// properties //bean
    private String name;
    private String city;

    //method creation
     void setAge(int a)
    {
        age = a;
    }
    int getAge()
    {
        return age;
    }
}

public  class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();
        st.setAge(40);
        int age = st.getAge();

        System.out.println(age);
    }
}
```

Controlled access to data members, by controlling direct access -by using data binding through setters and getters

How to achieve encapsulation in java?

- Using private keywords
- Using setters and getters

What is beans?

All variables are private.

```
private int age;
// instance variables //
//feeds// properties
private String name;
private String city;
```

Shadowing

```
{
    private int age;
    // instance variables // data members
    //feeds// properties
    private String name;
    private String city;

    //method creation
     void setAge(int age)
    {
        age = age;
    }
    int getAge()
    {
        return age;
    }


}

public  class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();
        st.setAge(40);
        int age = st.getAge();
        System.out.println(age);
    }
}
```

Output: 0

JVM is confused with same name

Shadowing problem

```
    //feeds// properties
    private String name;
    private String city;

    //method creation

    void setName(String name)
    {
        name = name;
    }
    String getName()
    {
        return name;
    }


public  class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();

        st.setName("rama");
        String name = st.getName();

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

Output: null

To overcome shadowing problem , we use this keyword

```java
class Student
{
    private int age;
    // instance variables // data members
    //feeds// properties
    private String name;
    private String city;
    //method creation
     void setAge(int age)
    {
        this.age = age;
    }
    int getAge()
    {
        return age;
    }
}

public  class Ecapsulation {
    public static void main(String[] args) {
        //new object creation -- instanciation
        Student st = new Student();
        st.setAge(40);
        int age = st.getAge();
        System.out.println(age);

    }
}
```

o/p → 40

Right click

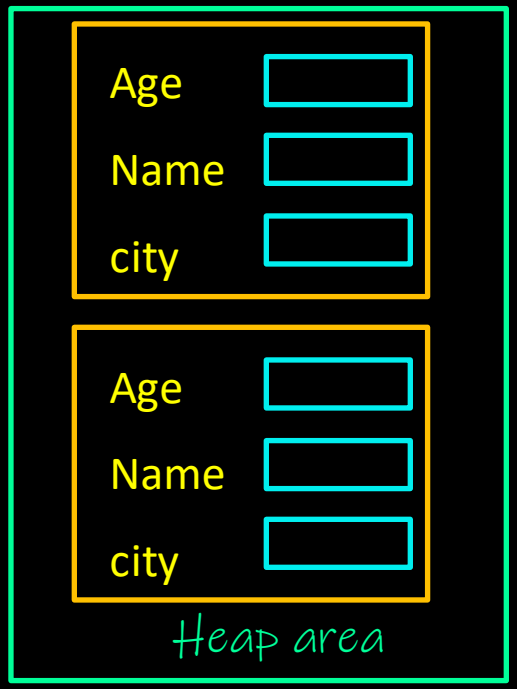
↓

source

↓

generate setter and getters

One class can create multiple objects



Ex.

```
Student st1 = new Student();
```

```
Student st = new Student();
```

```java
class Dog
{
    private String name;
    private int cost;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getCost() {
        return cost;
    }
    public void setCost(int cost) {
        this.cost = cost;
    }
}
public class EncapDog {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.setName("Sheru");
        d.setCost(2000);

        int cost = d.getCost();
        String name = d.getName();

        System.out.println("cost of "+  name + " is: " + cost);
    }
}
```

Two variables and 2 setters getter pairs

```java
class Student2
{
    private int age;
    private String name;
    private String city;
    public void setData(int age, String name,String city) {
        this.age = age;
        this.name = name;
        this.city = city;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }

    public String getCity() {
        return city;
    }
}
```

Common setters

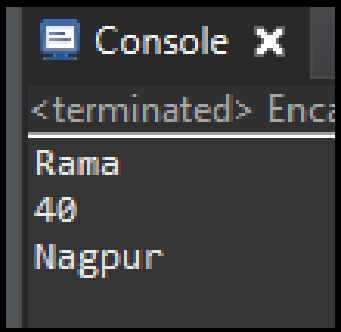
Not recommended

No common getters

```java
public class EncapStudent2 {
    public static void main(String[] args) {
        Student2 st = new Student2();

        st.setData(40,"rama","Nagpur");
        int age = st.getAge();
        String name = st.getName();
        String city = st.getCity();

        System.out.println(age);
        System.out.println(name);
        System.out.println(city);

    }

}
```

Constructor

```
class Student1
{
    private int age;
    // instance variables // data members
    //feeds// properties
    private String name;
    private String city;

    Student1(String name, int age, String city)
    {
        this.name= name;
        this.age = age;
        this.city  = city;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getCity() {
        return city;
    }
}
public class EncapStudent {

    public static void main(String[] args) {
    Student1 std1 = new Student1("Rama",40,"Nagpur");

    System.out.println(std1.getName());
    System.out.println(std1.getAge());
    System.out.println(std1.getCity());
    }

}
```



Constructor should have same name as the class

There can be number of parameters: Single or multiple parameters

Constructors doesn't have return type

```java
class Student1
{
    private int age;
    // instance variables // data members
    //feeds// properties
    private String name;
    private String city;

    Student1(String name, int age, String city)
    {
        this.name= name;
        this.age = age;
        this.city  = city;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getCity() {
        return city;
    }
    }
```

```java
public class EncapStudent {

    public static void main(String[] args) {
    Student1 std1 = new Student1("Rama",40,"Nagpur");

    System.out.println(std1.getName());
    System.out.println(std1.getAge());
    System.out.println(std1.getCity());
    }

}
```

Constructor is not the replacement of setters

Constructor is called automatically as you create an object, and pass values to variables

When you want to pass some value during object creation , what to do?
Keep that variables in the constructor

Constructor is called at the time of initialization

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}

public class encapstd1 {
    public static void main(String[] args) {
        Student3 std3=new Student3();
        std3.disp();

        System.out.println(std3.getAge());
        System.out.println(std3.getName());
        System.out.println(std3.getCity());

    }
}
```

<terminated> encapst

```
0
null
null
```

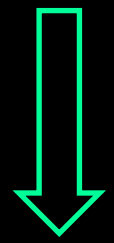
If you have not included any constructor , then JVM/JAVA complier will include java constructor behind the scenes, when you call a constructor

No variables so error

```
2  class Student3
3  {
4      private int age;
5      private String name;
6      private String city;
7
8      public int getAge() {
9          return age;
10     }
11     public String getName() {
12         return name;
13     }
14     public String getCity() {
15         return city;
16     }
17 }
18
19 public class encapstd1 {
20     public static void main(String[] args) {
21         Student3 std3=new Student3(40,"rama","Ngp");
22         std3.disp();
23
24         System.out.println(std3.getAge());
25         System.out.println(std3.getName());
26         System.out.println(std3.getCity());
27
28     }
29 }
30
```

JVM will include constructor, but the constructor will be zero parameter

As you have calked parameterized constructor.



So, error

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    Student3(int age,String name,String city)
    {
        this.age =age;
        this.name = name;
        this.city = city;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}
public class encapstd1 {
    public static void main(String[] args) {
        Student3 std3=new Student3(40,"rama","Ngp");
        System.out.println(std3.getAge());
        System.out.println(std3.getName());
        System.out.println(std3.getCity());
    }
}
```

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    Student3(int age,String name,String city)
    {
        this.age =age;
        this.name = name;
        this.city = city;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}
public class encapstd1 {
    public static void main(String[] args) {
        Student3 std3=new Student3(40,"rama","Ngp");
        System.out.println(std3.getAge());
        System.out.println(std3.getName());
        System.out.println(std3.getCity());

        Student3 std2 = new Student3();
    }
}
```

If there is call to any constructor and if and only if there is not constructor available, then only the JVM will include the default zero parametrized constructor

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    Student3(int age,String name,String city)
    {
        this.age =age;
        this.name = name;
        this.city = city;
    }
    Student3()
    {
        name="CHinni";
        age = 3;
        city= "xyz";
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}
public class encapstd1 {
    public static void main(String[] args) {
        Student3 std3=new Student3(40,"rama","Ngp");
        System.out.println(std3.getAge());
        System.out.println(std3.getName());
        System.out.println(std3.getCity());

        Student3 std2 = new Student3();
    }
}
```

Constructor included

As no error

Here student3 constructor is mentioned twice , so it is called constructor overloading

Constructor:

1. Special setter with same name as the class
2. No return type(explicit), called during object creation
3. Multiple constructor with different parameters can be created
4. It can have specifies like private and public
5. It does not have return type
6. JVM will create default constructor , if there is a call for constructor and no constructor is available
7. Constructor created by JVM is zero parameterized type

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    Student3(int age,String name,String city)
    {
        this.age =age;
        this.name = name;
        this.city = city;
    }
    Student3()
    {
        age = 3;
        city= "xyz";
    }
    Student3(String name)
    {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}
```

```java
public class encapstd1 {
    public static void main(String[] args) {
        Student3 std3=new Student3(40,"rama","Ngp");
        System.out.println(std3.getAge());
        System.out.println(std3.getName());
        System.out.println(std3.getCity());

        Student3 std2 = new Student3();
        System.out.println(std2.getAge());
        System.out.println(std2.getName());
        System.out.println(std2.getCity()   );
    }
}
```

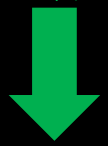
Console

<terminated> encapstd1 [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin

```
40
rama
Ngp
3
null
xyz
```

```java
class Student3
{
    private int age;
    private String name;
    private String city;

    Student3(int age,String name,String city)
    {     Super()
        this.age =age;
        this.name = name;
        this.city = city;
    }
    Student3()
    {
        age = 3;
        city= "xyz";
    }
    Student3(String name)
    {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public String getCity() {
        return city;
    }
}
```

Super method will be in topline of the constructor body (behind the scenes)



Call parent type

Super() and this() methods don't coexist as they exist in 1st line only

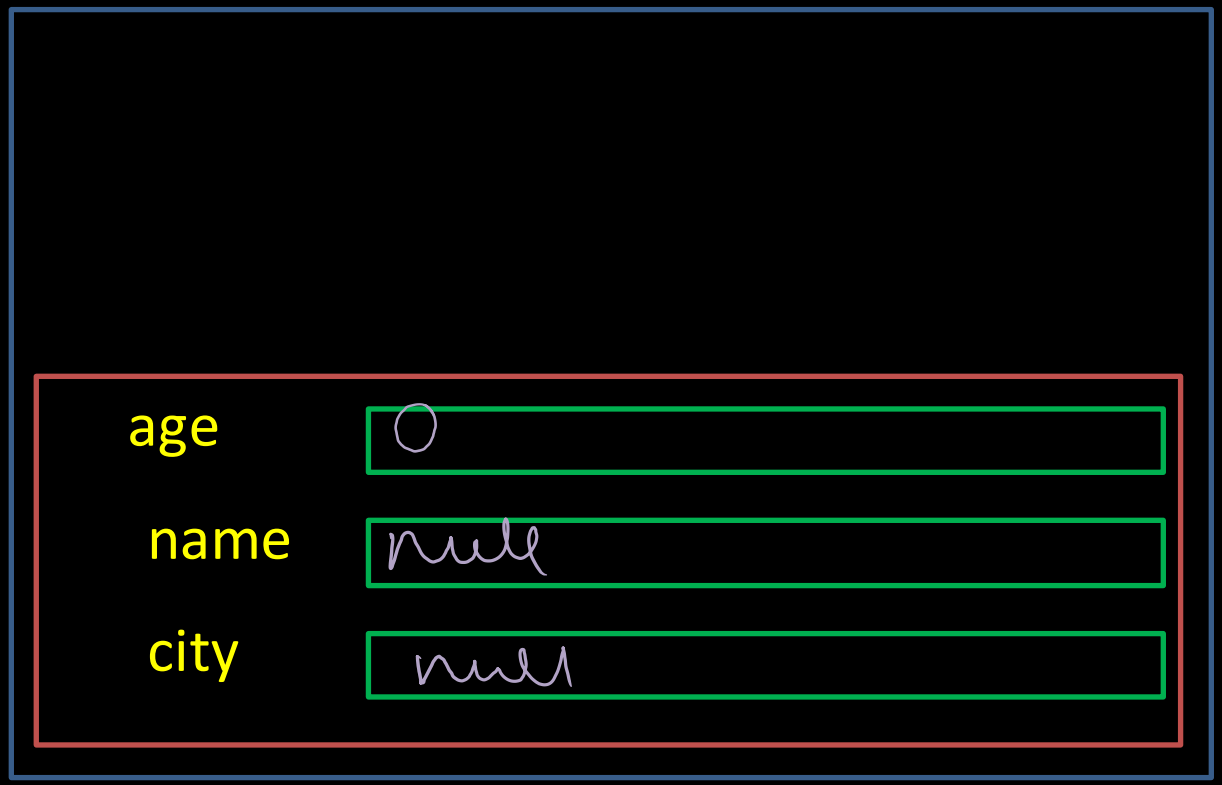
Super(): will call parent class constructor

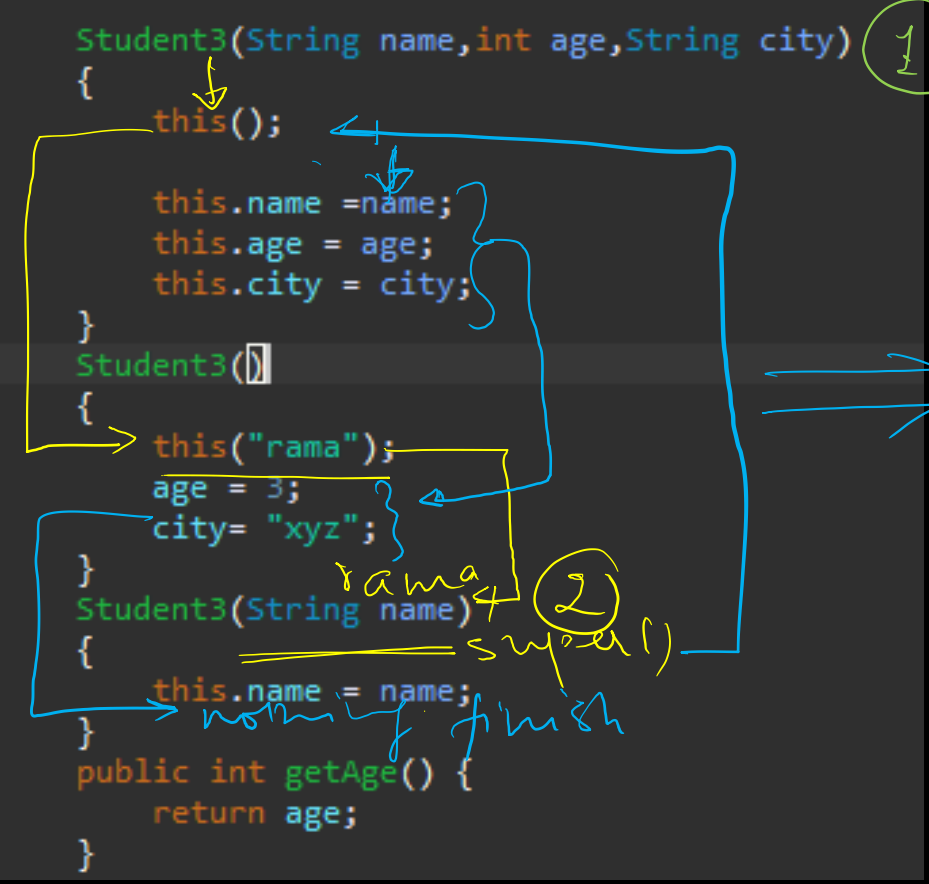
this( ) : will call constructor of same class

```
8     Student3(int age,String name,String city)
9     {
10        this();
11        super();
12        this.age =age;
13        this.name = name;
14        this.city = city;
15    }
```

```
9     {
10        super();
11        this();
12
13        this.age =age;
14        this.name = name;
15        this.city = city;
16    }
```

```java
Student3(int age,String name,String city)
{
    this();

    this.age =age;
    this.name = name;
    this.city = city;
}
```



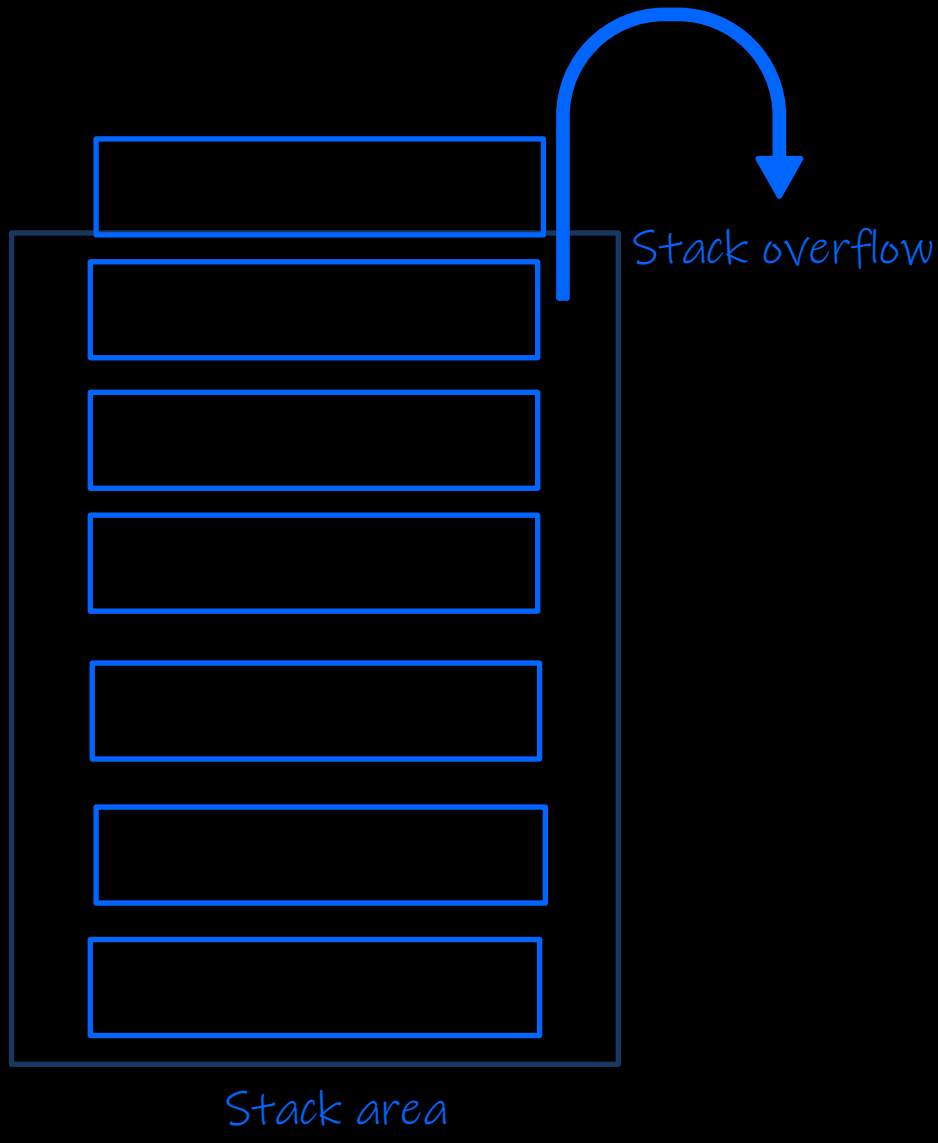


Constructor chaining

Moving between child class-

child class and to parent class

stack overflow



Will continue till the stack area is full, this is known as stack overflow

```java
    Student3(String name,int age,String city)
    {
        this("charan", 40, "Vizag");//stack overflow

        this.name =name;
        this.age = age;
        this.city = city;
    }
    Student3()
    {
        this("rama");
        age = 3;
        city= "xyz";
    }
    Student3(String name)
    {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
```

Continuous loop so stack overflow.

Not recommended

Not excepting integer value
Only string values are excepted

```java
    Student3(String name,int age,String city)
    {
        this( 40);//not excepting integer value

        this.name =name;
        this.age = age;
        this.city = city;
    }
    Student3()
    {
        this("rama");
        age = 3;
        city= "xyz";
    }
    Student3(String name)
    {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
```

Conclusion

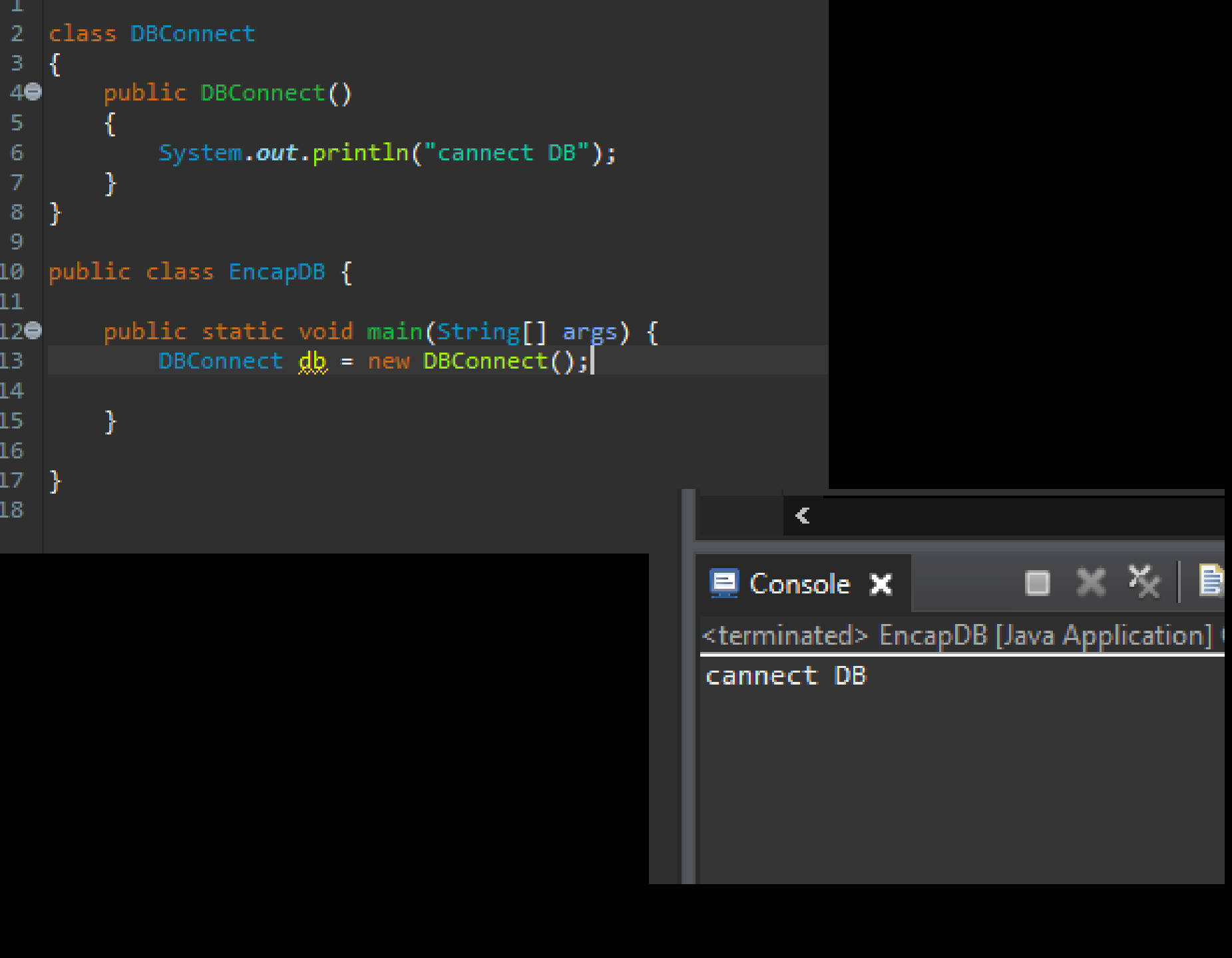
1. Constructor has same name as that of class
2. It does not have return values
3. Can't write return statement inside a constructor
4. We can have more than one constructor
5. Even if we have more that one constructor , they should have different parameters
6. Constructor get invoked when we initiate / create an object
7. This() -→constructor of same class
8. Super() → calls parents type constructor
9. TVM will include default constructor if programmer has not specified any constructor
10. Constructor calling constructor is called constructor chaining
11. If some statement has to be executed the moment, we create objects- we can write that inside a constructor

This	This( )
• Keyword • Refer to current object	• method • It will call some class constructor

Constructor overloading:

More than one constructor with different parameters is called constructor overloading

methods	constructor
1. Call explicitly by calling name	1. Called when object is created
2. Return type explicitly (void , int, string)	2. No explicit return type
3. Return statement allowed	3. Return statement not allowed


class DBConnect
{
public DBConnect()
{
System.out.println("cannect DB");
}
}
public class EncapDB {
public static void main(String[] args) {
DBConnect db = new DBConnect();
}
}
Console
<terminated> EncapDB [Java Application]
cannect DB