Outline

- Basic Concepts
  - Defining Object
  - Defining is class
  - Construction
  - Method

Object and Class

What is Class?

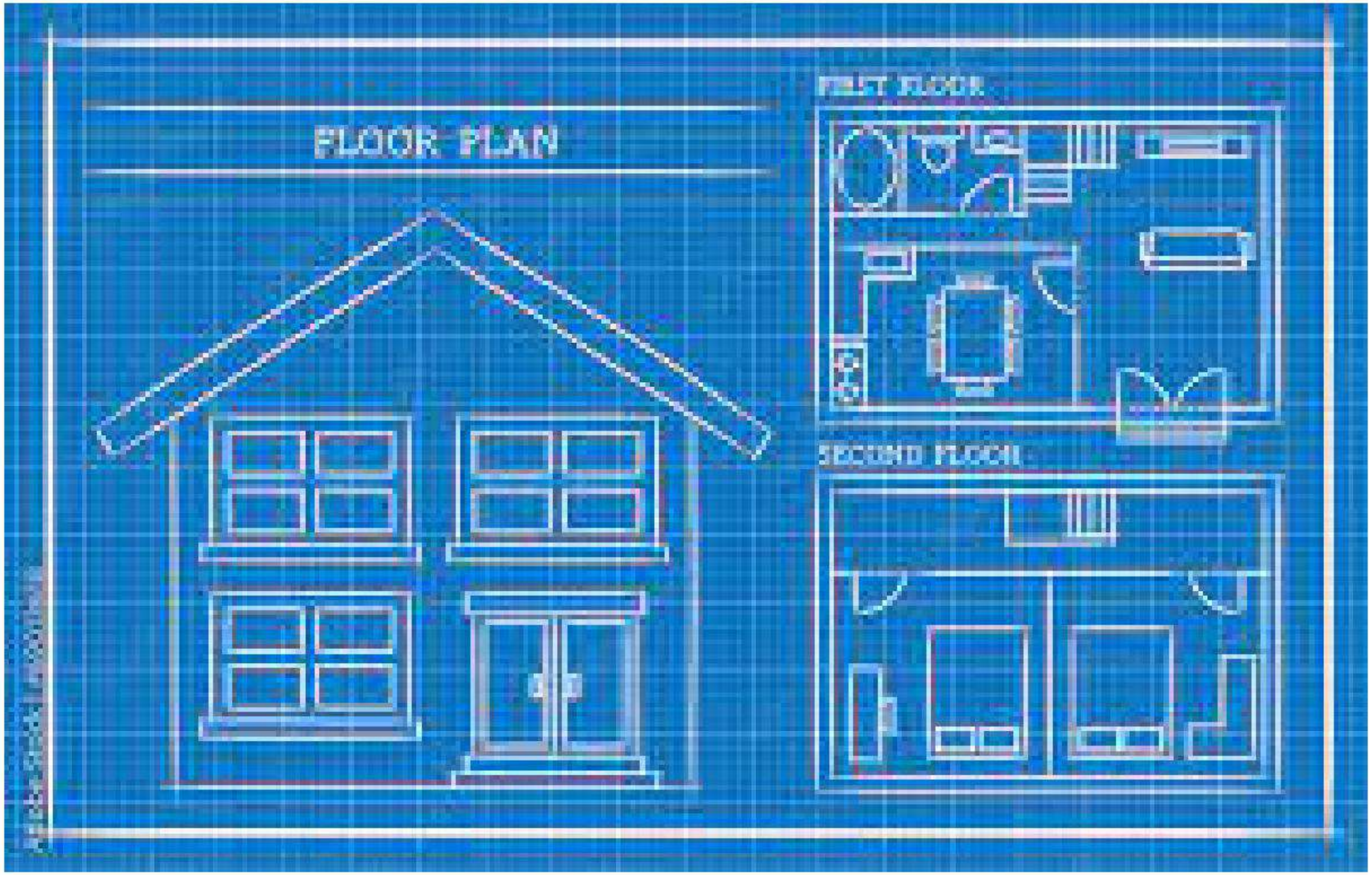
- In object-oriented programming, **a class** is **a basic building block.**
- A class is a **blueprint** or **prototype** that defines the **variables** and **methods** common to **all objects of a certain kind**
- A class is **a group of objects** which **have common properties**.
- **Before** we create **an object**, we **first** need to **define the class**.
- It is **a logical entity**.
  - It can't be physical

Object and Class

What is Class?

- We can think of the **class as prototype of a house.**
  - It contains **all the details** about **the floors, doors, windows, etc.**
- Based on these **descriptions we build the house.**
- Since **many houses** can be **made from the same description**
  - We can **create many objects from a class.**

Object and Class

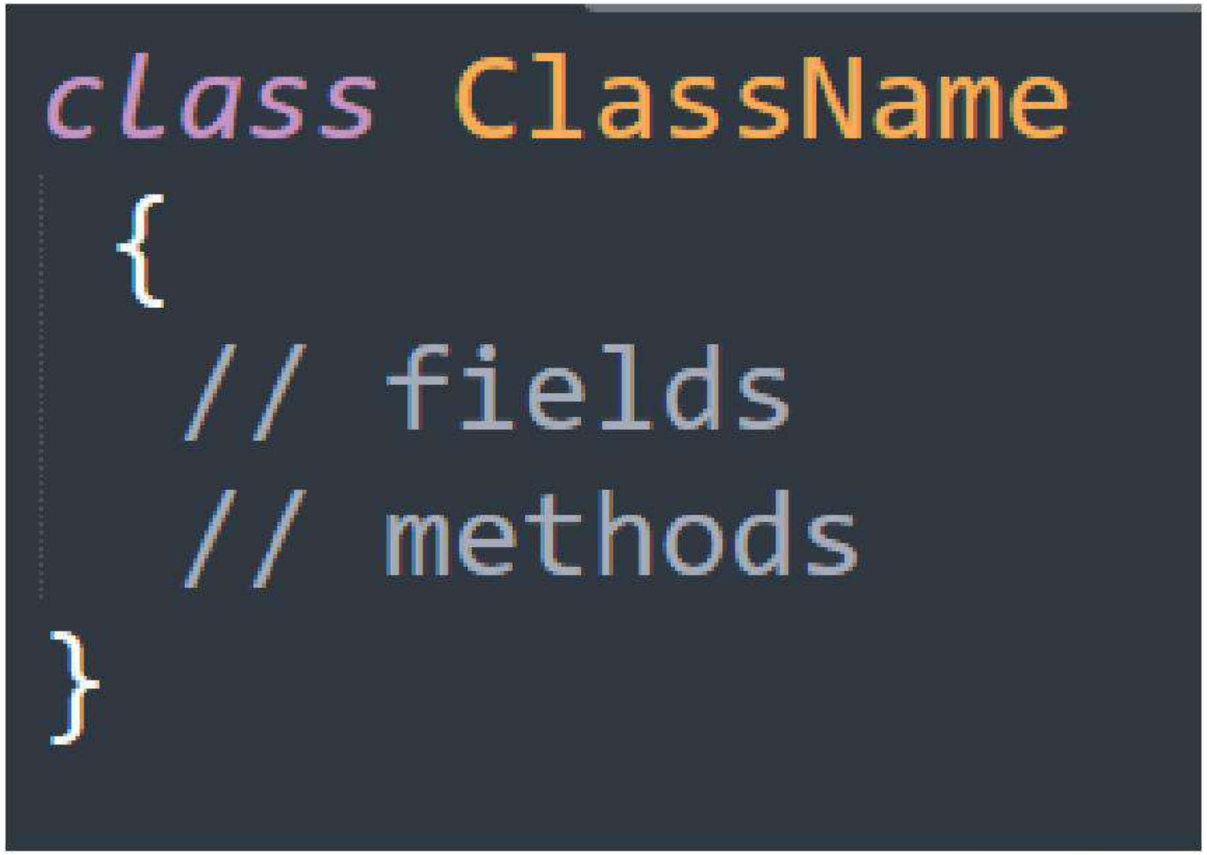


-

Object and Class

What is Class?

- We can **create a class** in Java using the **class keyword.**
- Class is **a group of variables** of **different data types** and **group of methods.**
- **Syntax to declare a class:**

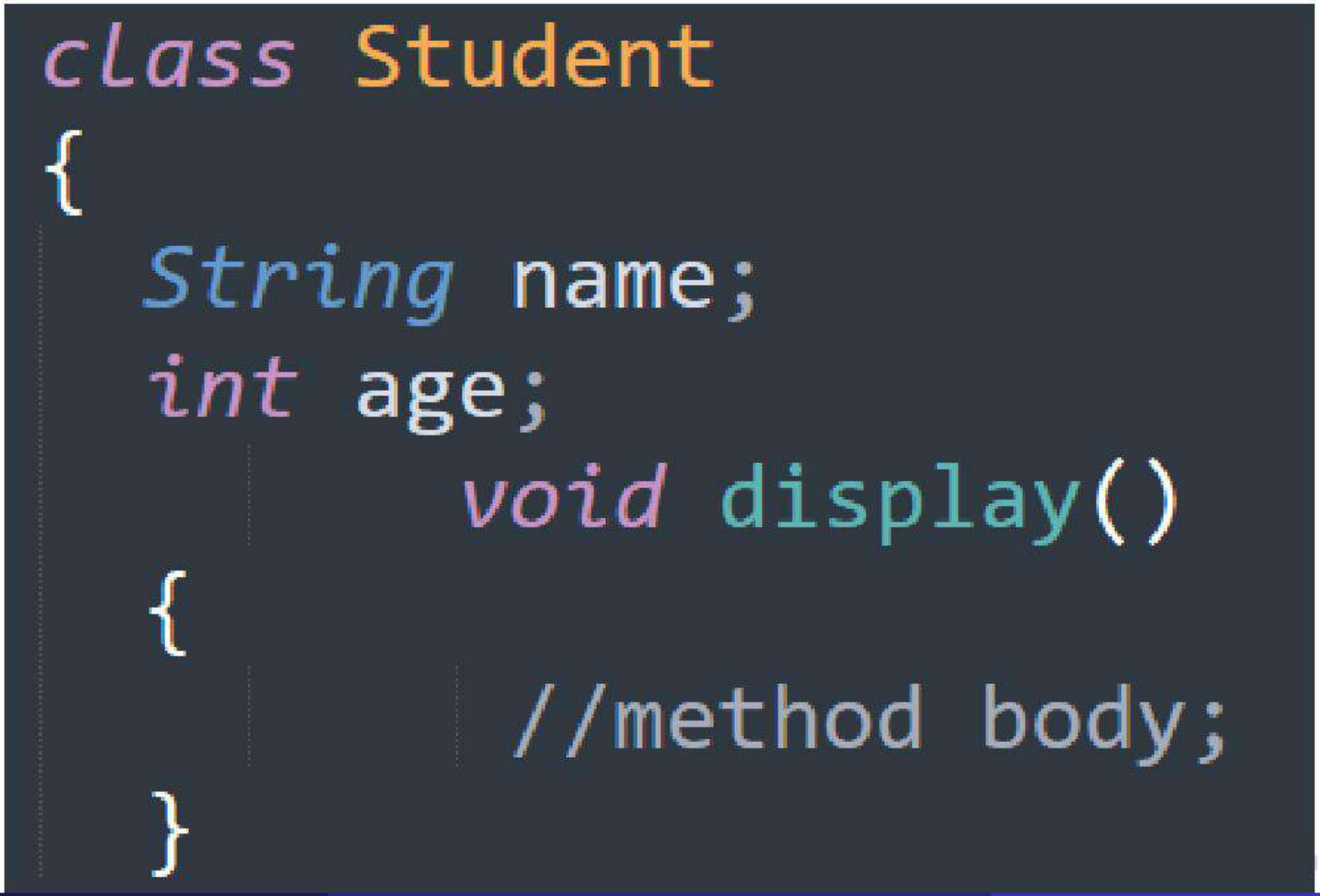


```
class ClassName
 {
  // fields
  // methods
}
```

Object and Class

What is Class?

- **fields (variables) and methods represent** the **state** and **behavior** of the **object** respectively.
  - fields are used to store data
  - methods are used to perform some operations



```
class Student
{
  String name;
  int age;
          void display()
  {
           //method body;
  }
```

Object and Class

What is Class?

- The **data, or variables**, **defined within a class are called instance variables.**
- The **methods and variables** defined within a class are called **members of the class.**

Object and Class

What is Object?

- **An object** is an identifiable entity with **some characteristics**, **state** and **behavior**.
- **An object** is called **an instance of a class.**
- It is a basic unit of Object-Oriented Programming and represents real life entities.
- A typical Java program creates many objects, which as you know, interact by invoking methods.

Object and Class

-

Object and Class

What is Object?

- An object has three characteristics:
  1. **State**: represents the **data (value) of an object.**
     - ★ item What does it look like?
  2. **Behavior**: represents **the behavior (functionality) of an object**
     - ★ What does it do?
  3. **Identity**: An object identity is typically implemented using **a unique ID.**
     - ★ What do we call it?

Object

For Example

- Pen is an object.
  - Its **name** is **Reynolds**;
  - **color** is **white**, known as its **state**.
  - It is **used to write**, so **writing is its behavior.**

Object

For Example

- To **create an object of a class**, **specify the class name**, **followed by the object name**, by using the **new keyword**

```
ClassName objectName = new ClassName();
```

- **For Examples**

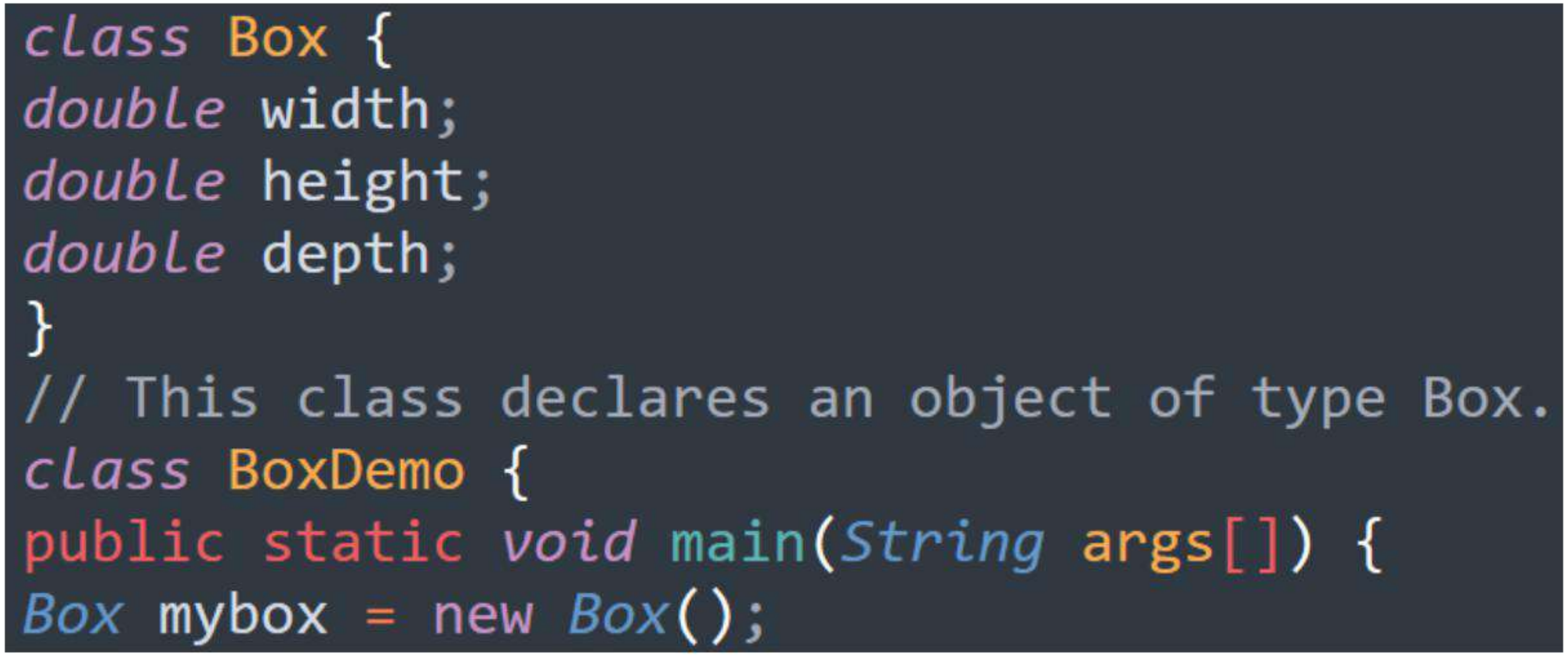
```
// for fruits class
fruits banana = new fruits();
fruits orange = new Bicycle();
```

Object

- The **new** operator **dynamically allocates** (that is, allocates at run time) **memory for an object**.
- **for Example:**
  Box mybox = new Box();
- This statement combines the **two steps**
  1. The first line declares **mybox as a reference to an object of type Box.**
     - ⋆ mybox contains the **value null**
  2. **The next line allocates an actual object** and **assigns a reference to it to mybox.**

Object

Examples



```
class Box {
double width;
double height;
double depth;
}
// This class declares an object of type Box.
class BoxDemo {
public static void main(String args[]) {
Box mybox = new Box();
```

-

Object

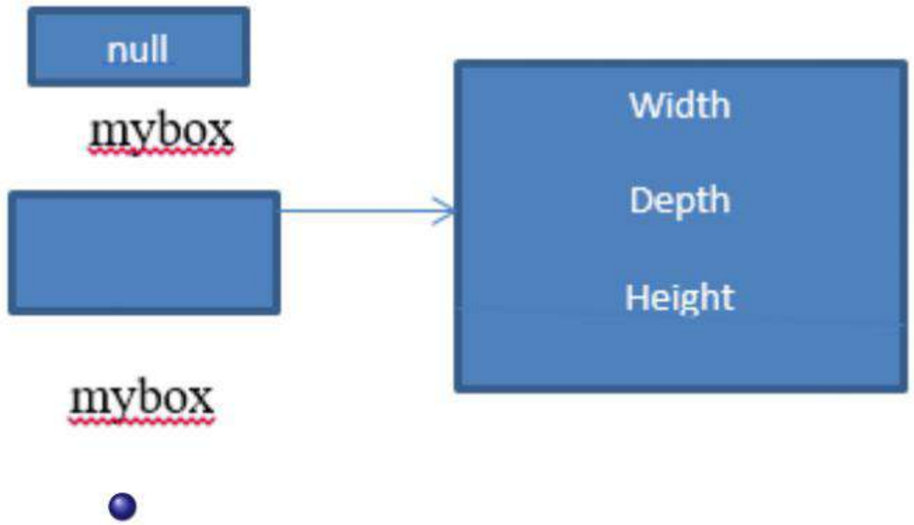
Examples

Statement

box mybox ;

mybox =new box();

Effect

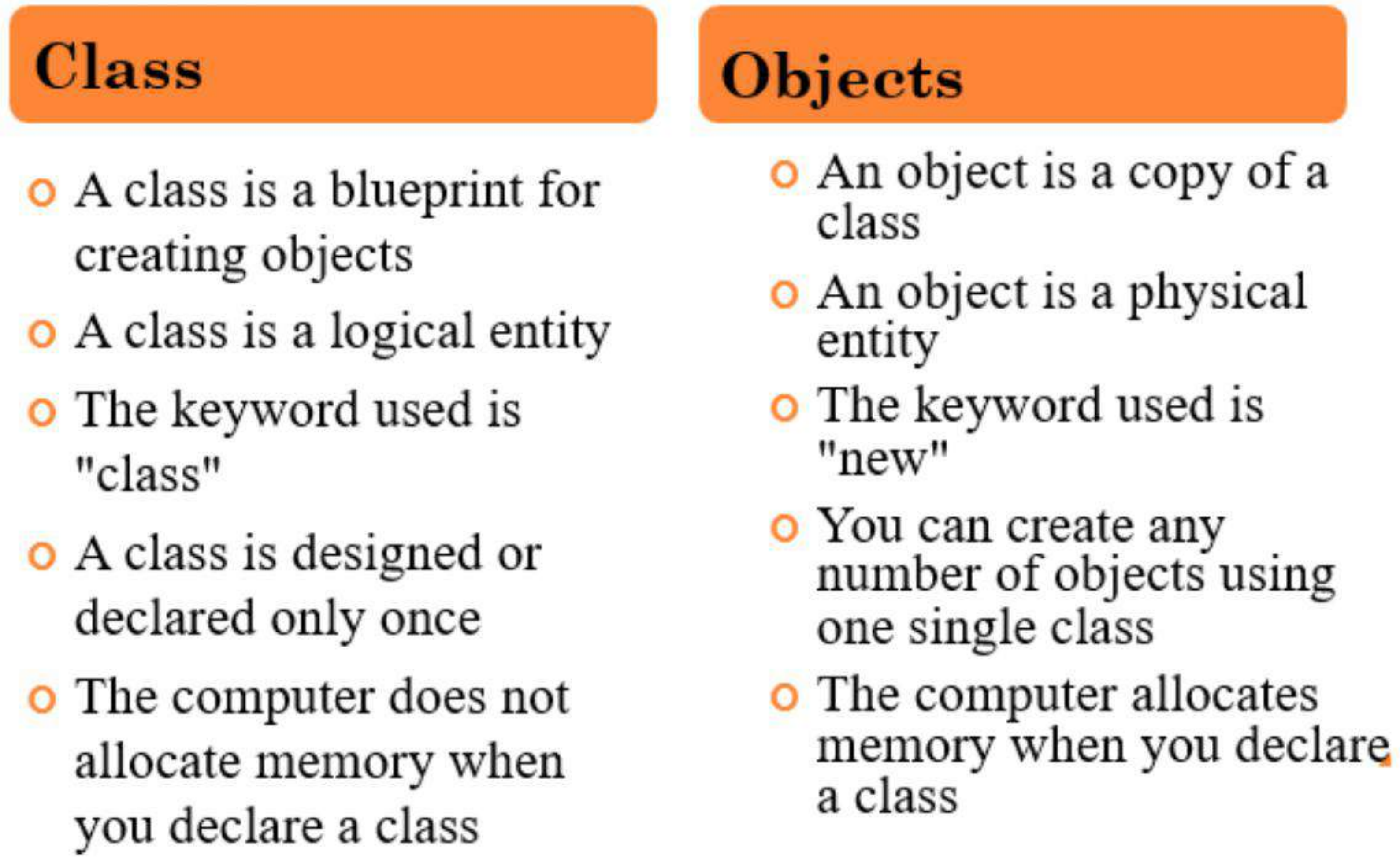


Object

- We can use the **name of objects** along with the **dot (.) operator** to **access members of a class.**
- The **dot operator links** the **name of the object** with **the name of an instance variable**.

Object

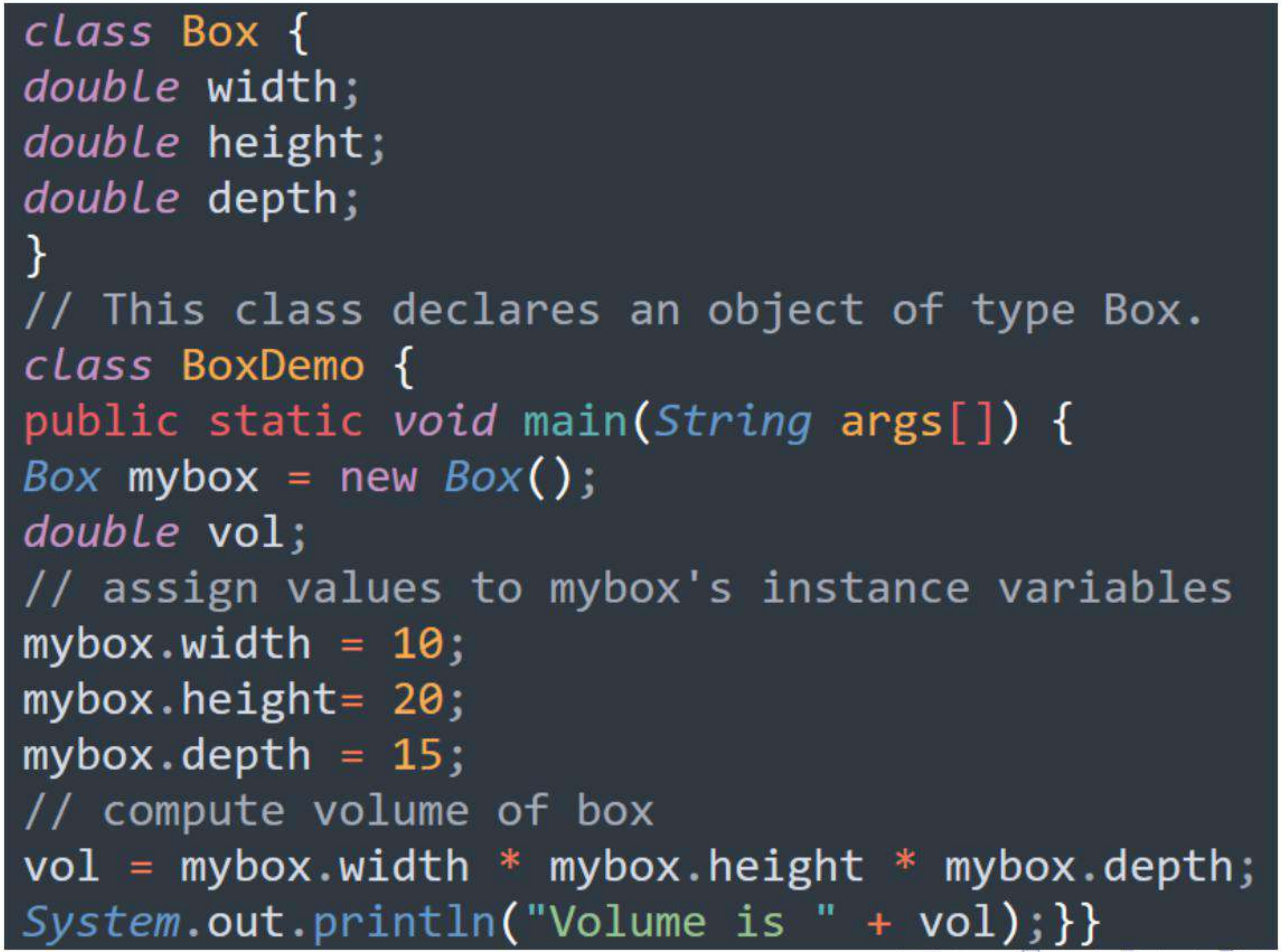
Key Differences Between Java Classes and Objects



Class	Objects
A class is a blueprint for creating objects	An object is a copy of a class
A class is a logical entity	An object is a physical entity
The keyword used is "class"	The keyword used is "new"
A class is designed or declared only once	You can create any number of objects using one single class
The computer does not allocate memory when you declare a class	The computer allocates memory when you declare a class

Object and Class

Example 1



```
class Box {
double width;
double height;
double depth;
}
// This class declares an object of type Box.
class BoxDemo {
public static void main(String args[]) {
Box mybox = new Box();
double vol;
// assign values to mybox's instance variables
mybox.width = 10;
mybox.height= 20;
mybox.depth = 15;
// compute volume of box
vol = mybox.width * mybox.height * mybox.depth;
System.out.println("Volume is " + vol);}}
```

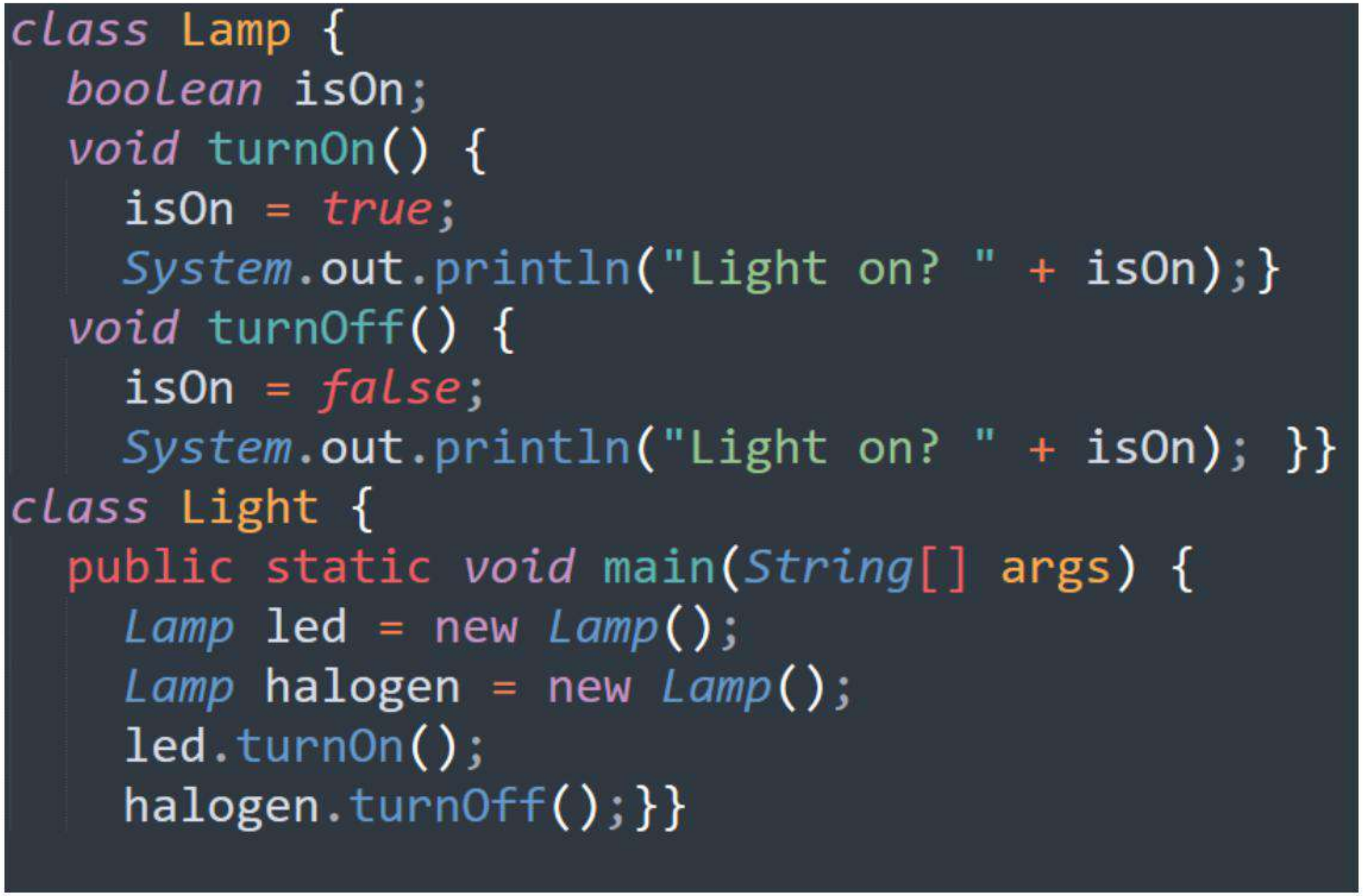
Object and Class

Example 1

- The output of Example one is:
  Volume is 3000.0

Object and Class

Example 2



```
class Lamp {
  boolean isOn;
  void turnOn() {
    isOn = true;
    System.out.println("Light on? " + isOn);}
  void turnOff() {
    isOn = false;
    System.out.println("Light on? " + isOn); }}
class Light {
  public static void main(String[] args) {
    Lamp led = new Lamp();
    Lamp halogen = new Lamp();
    led.turnOn();
    halogen.turnOff();}}
```

Object and Class

Example 2

- The output of Example Two is:

```
Light on? true
Light on? false
```

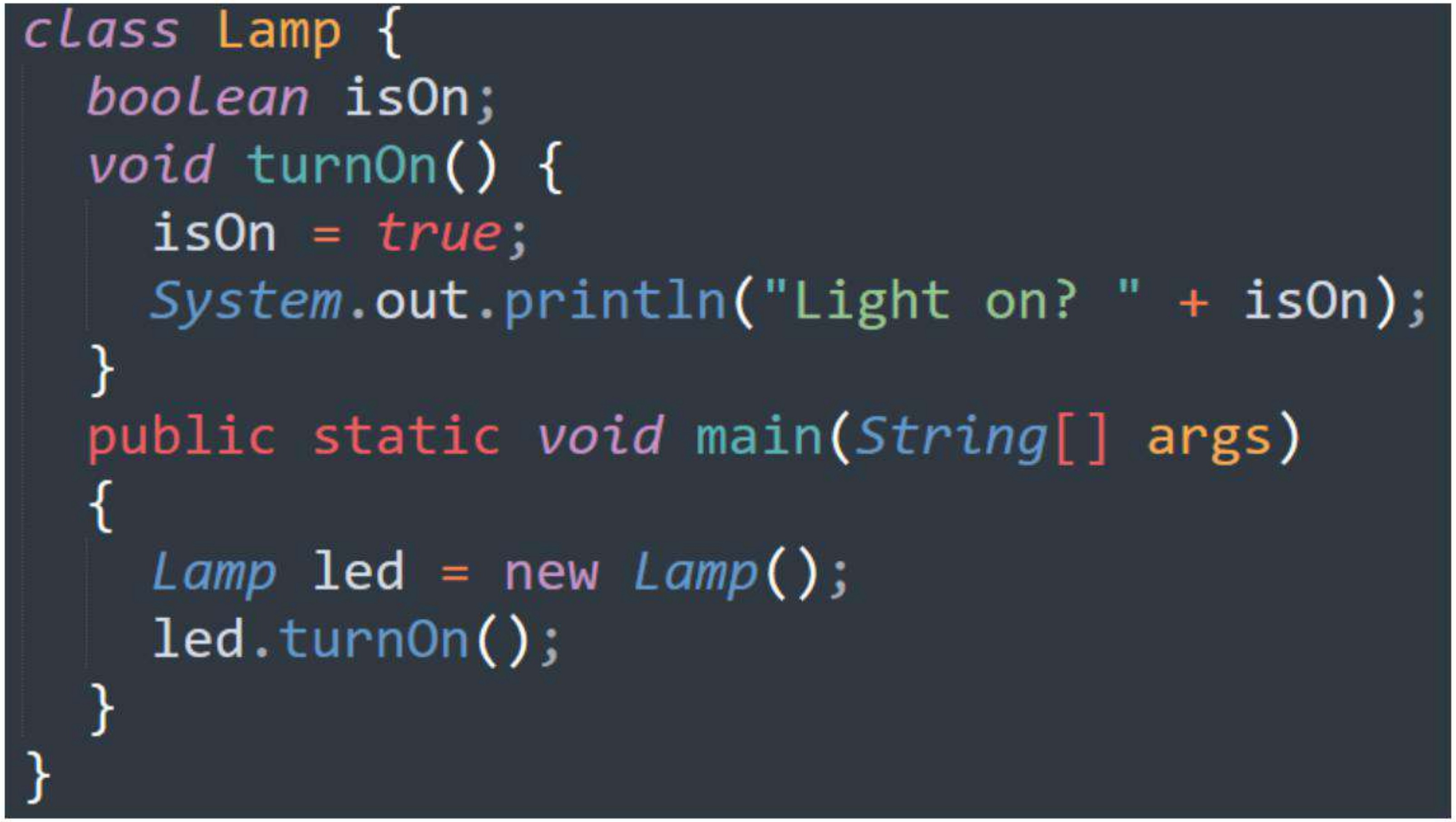
Object and Class

Create objects inside the same class

- **Note** that in the **previous example**, we have **created objects inside another class and accessed the members from that class.**
- **However**, we can also **create objects inside the same class.**

Object and Class

Example 3



```
class Lamp {
  boolean isOn;
  void turnOn() {
    isOn = true;
    System.out.println("Light on? " + isOn);
  }
  public static void main(String[] args)
  {
    Lamp led = new Lamp();
    led.turnOn();
  }
}
```

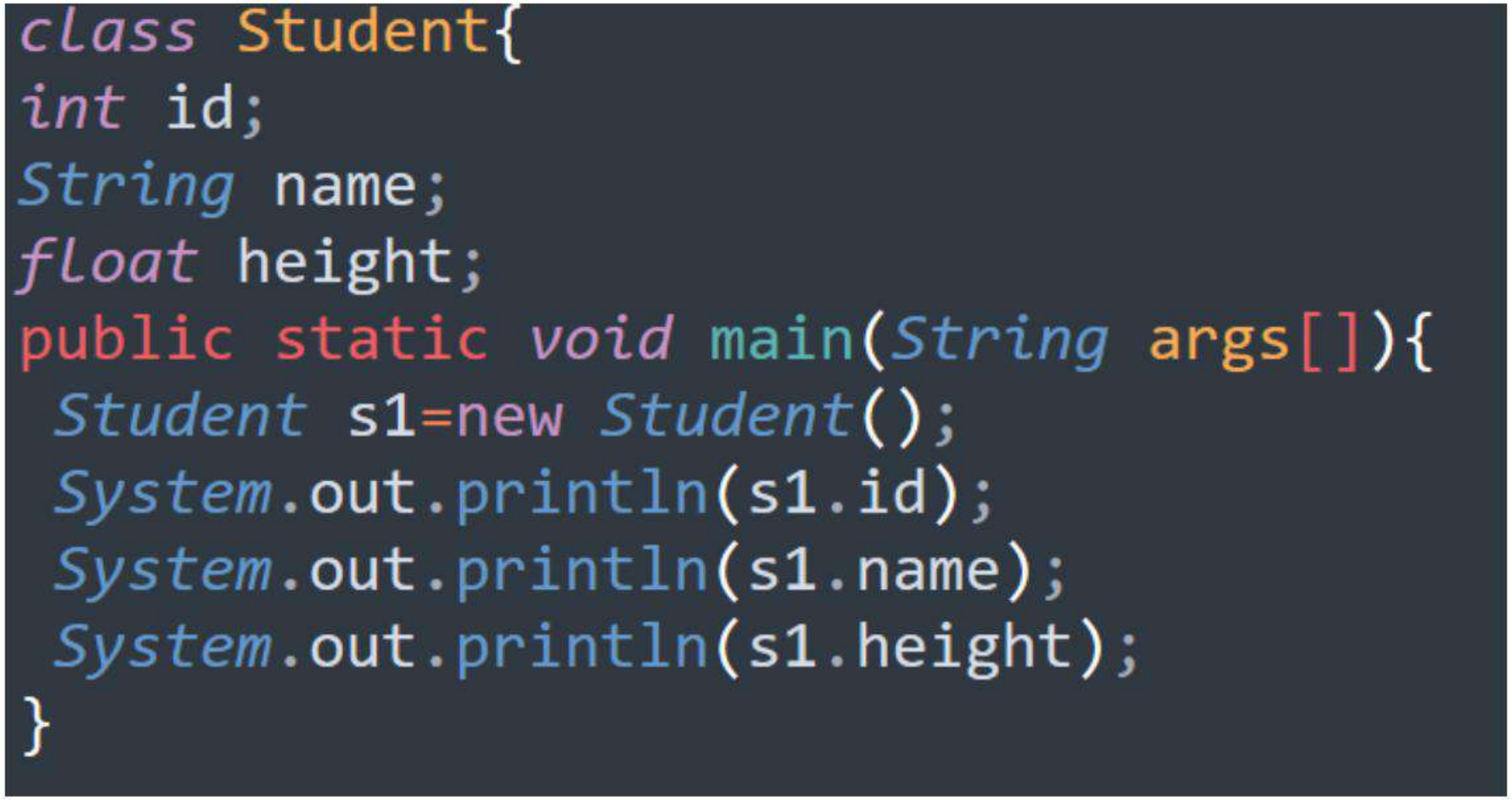
Object and Class

Example 3

- The output of Example Three is:
  Light on? true

Object and Class

Example 4



```
class Student{
int id;
String name;
float height;
public static void main(String args[]){
 Student s1=new Student();
 System.out.println(s1.id);
 System.out.println(s1.name);
 System.out.println(s1.height);
}
```

•

Object and Class

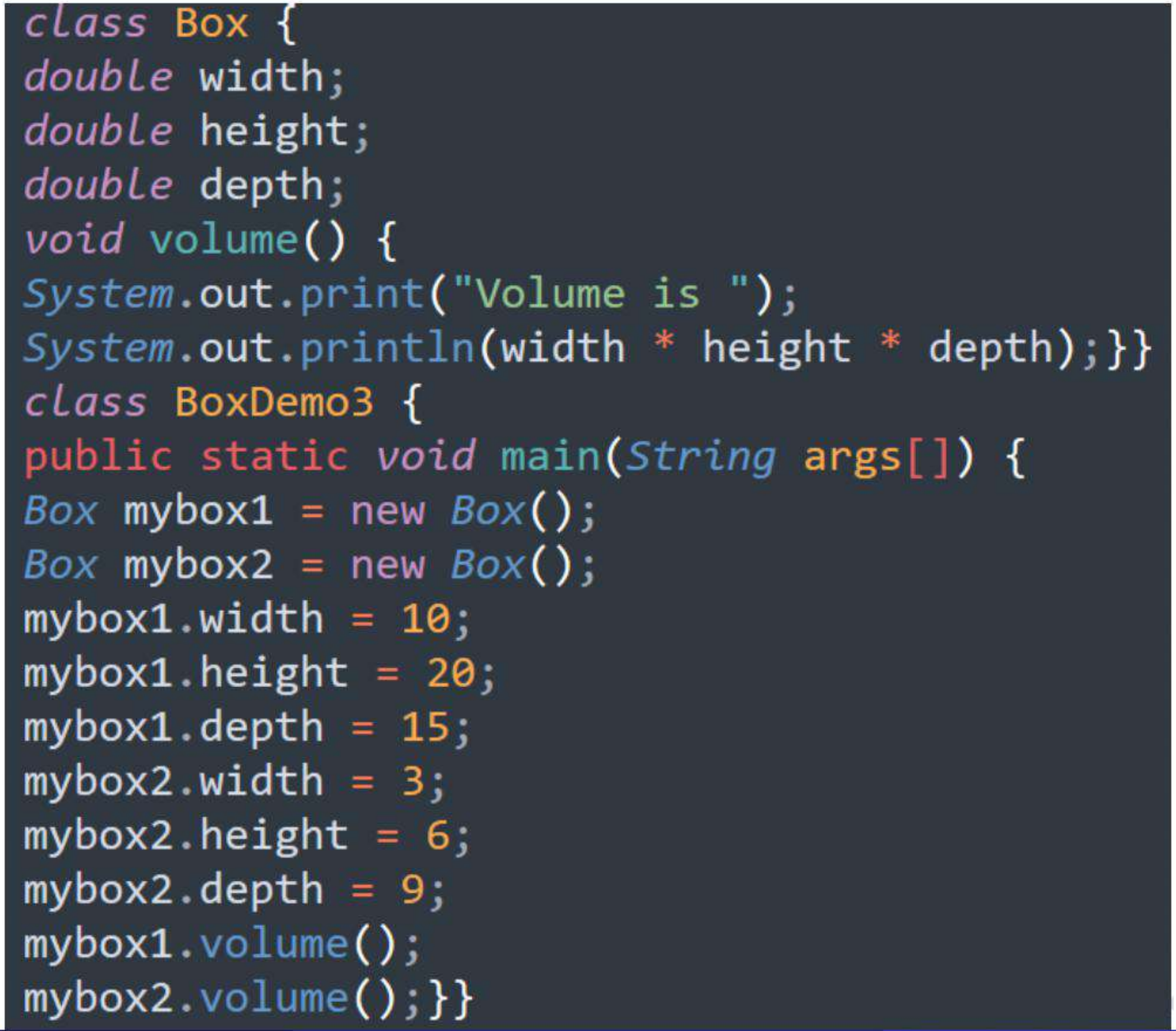
Example 4

- The output of Example Four is:

```
0
null
0.0
```

Object and Class

Example 5



```
class Box {
double width;
double height;
double depth;
void volume() {
System.out.print("Volume is ");
System.out.println(width * height * depth);}}
class BoxDemo3 {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box();
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
mybox1.volume();
mybox2.volume();}}
```

Object and Class

Example 5

- The output of Example Five is:

```
Volume is 3000.0
Volume is 162.0
```

Object and Class

Class Work

- Write the program that display the detail information about Employee
  .

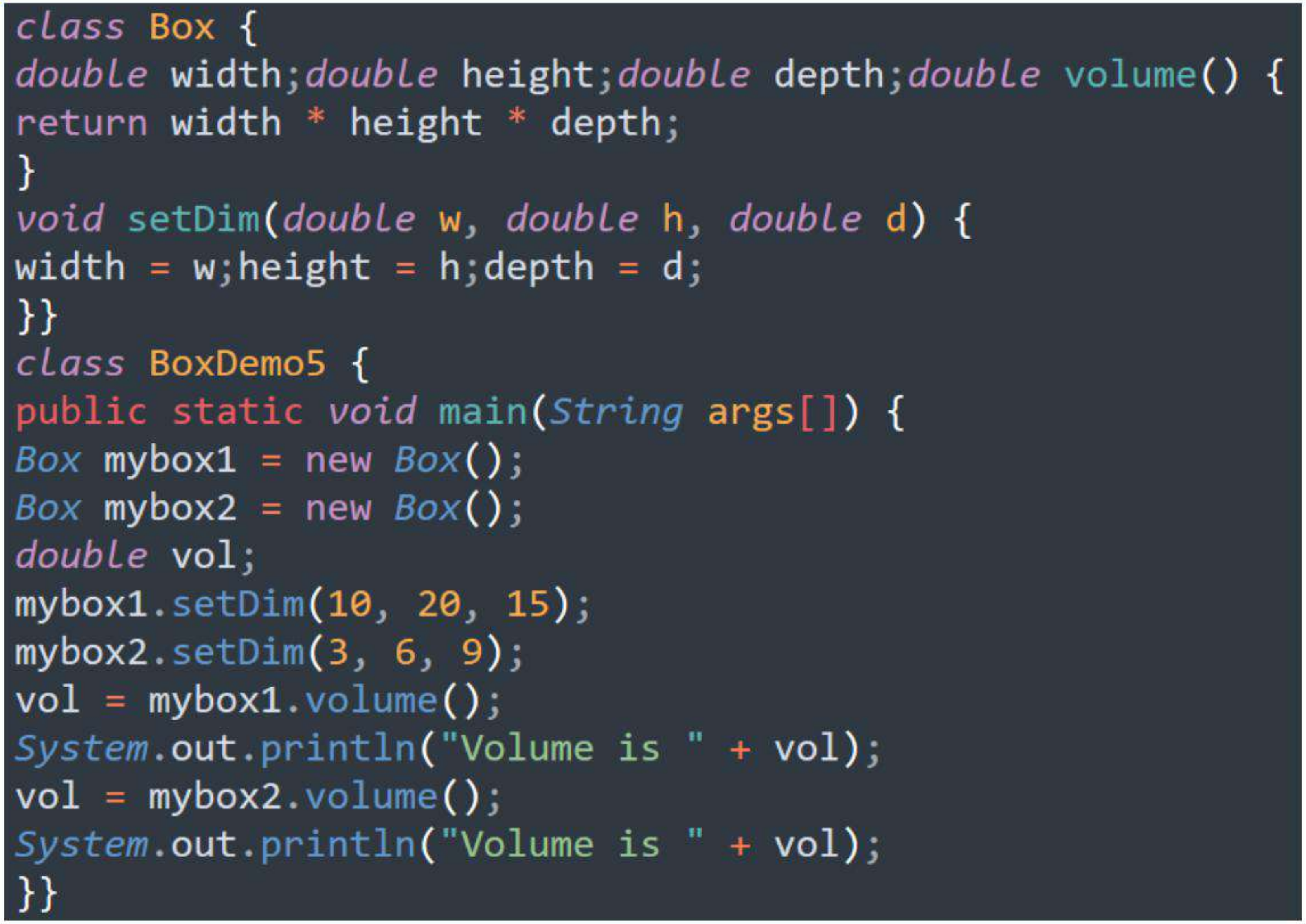
Object and Class

Adding a Method That Takes Parameters

- Some methods **don’t need parameters**, most do.
- Parameters allow a method to be generalized.

Object and Class

Adding a Method That Takes Parameters



```
class Box {
double width;double height;double depth;double volume() {
return width * height * depth;
}
void setDim(double w, double h, double d) {
width = w;height = h;depth = d;
}}
class BoxDemo5 {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
mybox1.setDim(10, 20, 15);
mybox2.setDim(3, 6, 9);
vol = mybox1.volume();
System.out.println("Volume is " + vol);
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}}
```

Object and Class

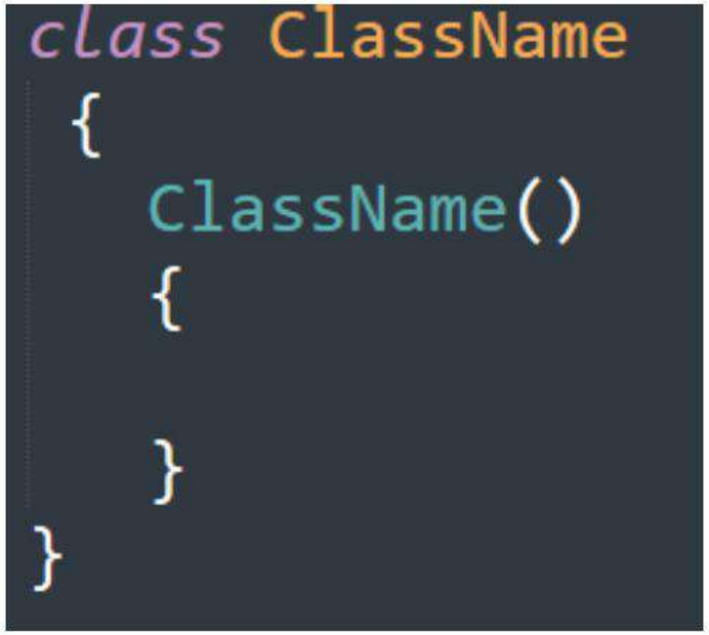
Constructors

- **A constructor** in Java is **a special method** that is used to **initialize objects.**
- The constructor is **called** when an **object of a class** is **created**.
- Constructor is **a block of code** that initializes the **newly created object.**
- It is called when an instance of the class is **created**.
  - At the time of **calling constructor**, **memory** for the **object is allocated in the memory**.

Object and Class

Constructors

- **Every time an object is created** using the **new() keyword, at least one constructor is called**.
- It is called constructor because it **constructs the values at the time of object creation.**
- Following is the **syntax of a constructor**



```
class ClassName
 {
   ClassName()
   {

   }
}
```

Object and Class

Rules for creating Java constructor

1. **A constructor** must have **the same name as the class itself.**
2. Constructors **do not** have **a return type—not even void.**

Object and Class

Types of Java constructors

- There are **Three types** of constructors in Java:
  1. **Default constructor**
  2. **No-arg constructor**
  3. **Parameterized.**

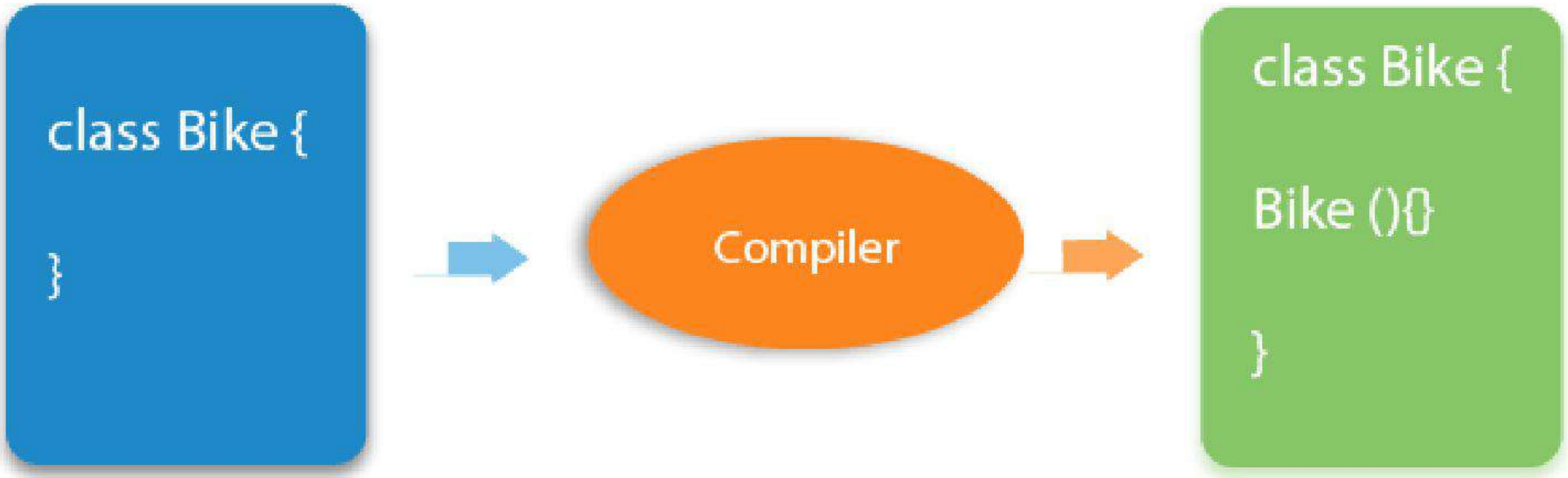
Types of Java constructors

Default constructor

- If you **do not implement** any **constructor in your class**, Java **compiler inserts a default constructor into your code**
- A default constructor **is invisible constructor.**
- **if we write a constructor with arguments or no arguments** then the compiler does not create a default constructor.

Types of Java constructors

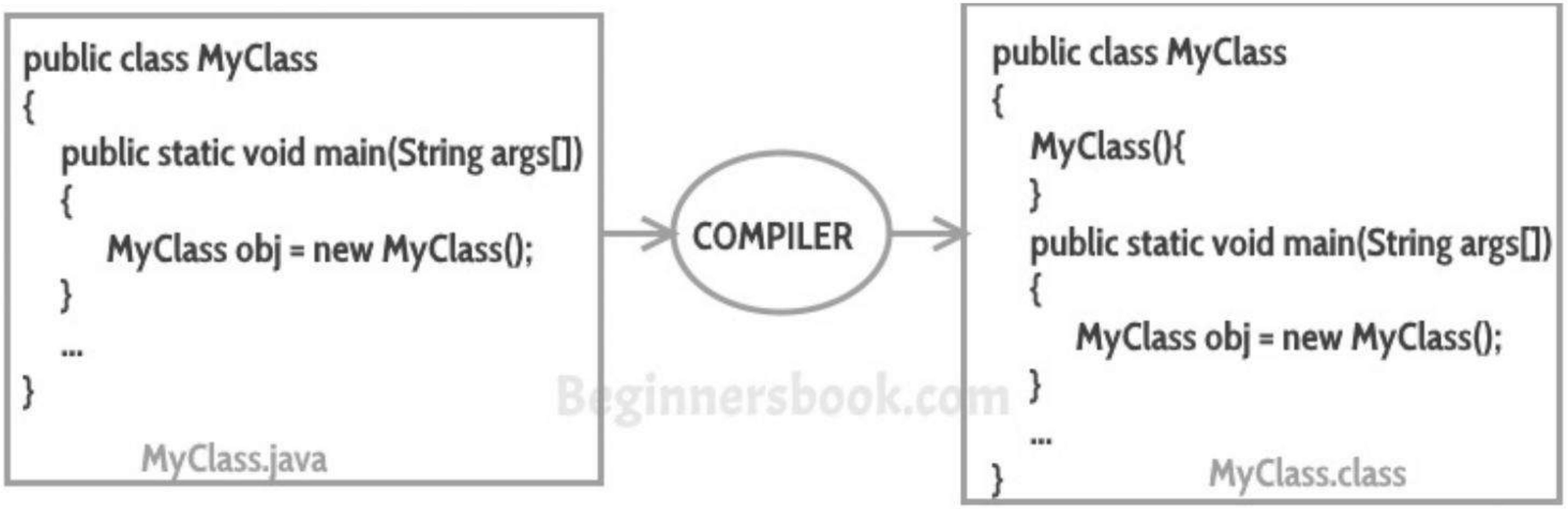
Default constructor



-

Types of Java constructors

Default constructor



```
public class MyClass
{
   public static void main(String args[])
   {
      MyClass obj = new MyClass();
   }
   ...
}
```

```
public class MyClass
{
   MyClass(){
   }
   public static void main(String args[])
   {
      MyClass obj = new MyClass();
   }
   ...
}
```

•

Types of Java constructors

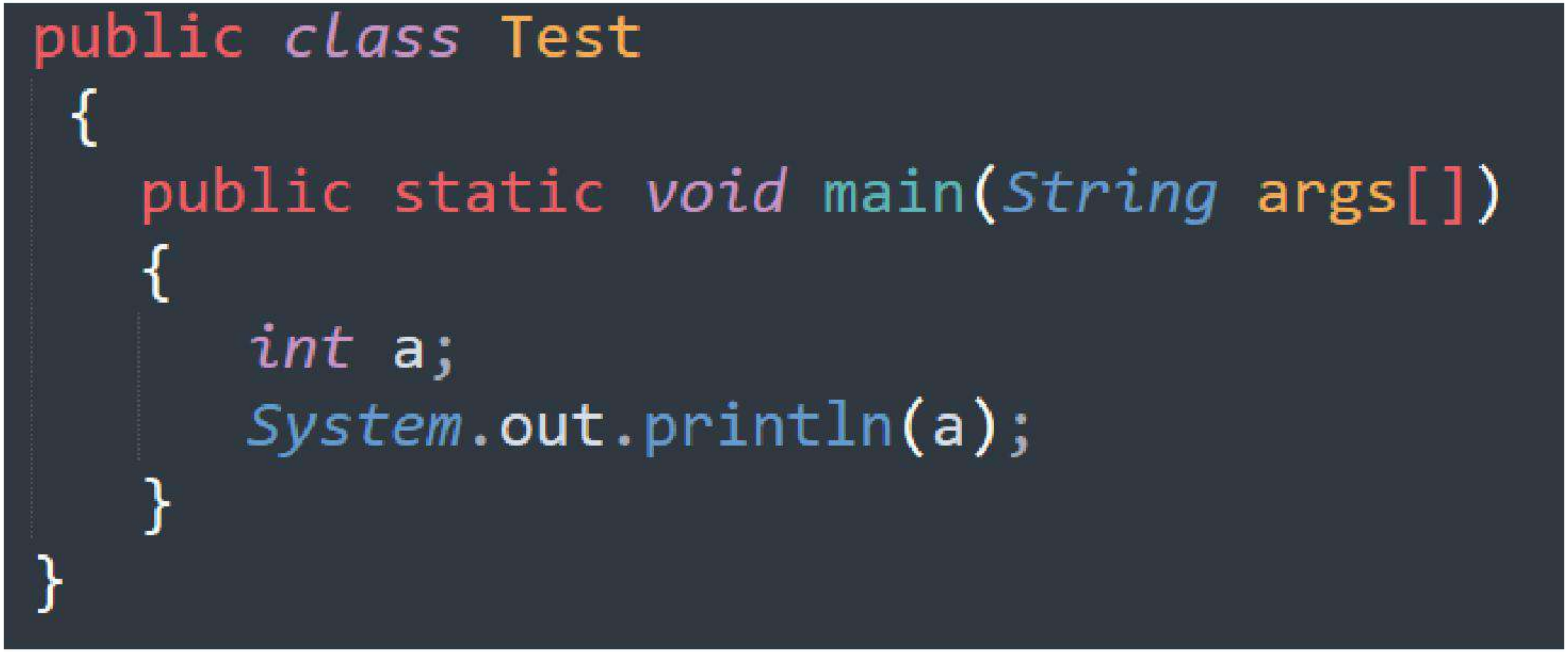
Class Work Question

- **What is the purpose of a default constructor?**

Types of Java constructors

Class Work Question

- **what is the output of the following code**

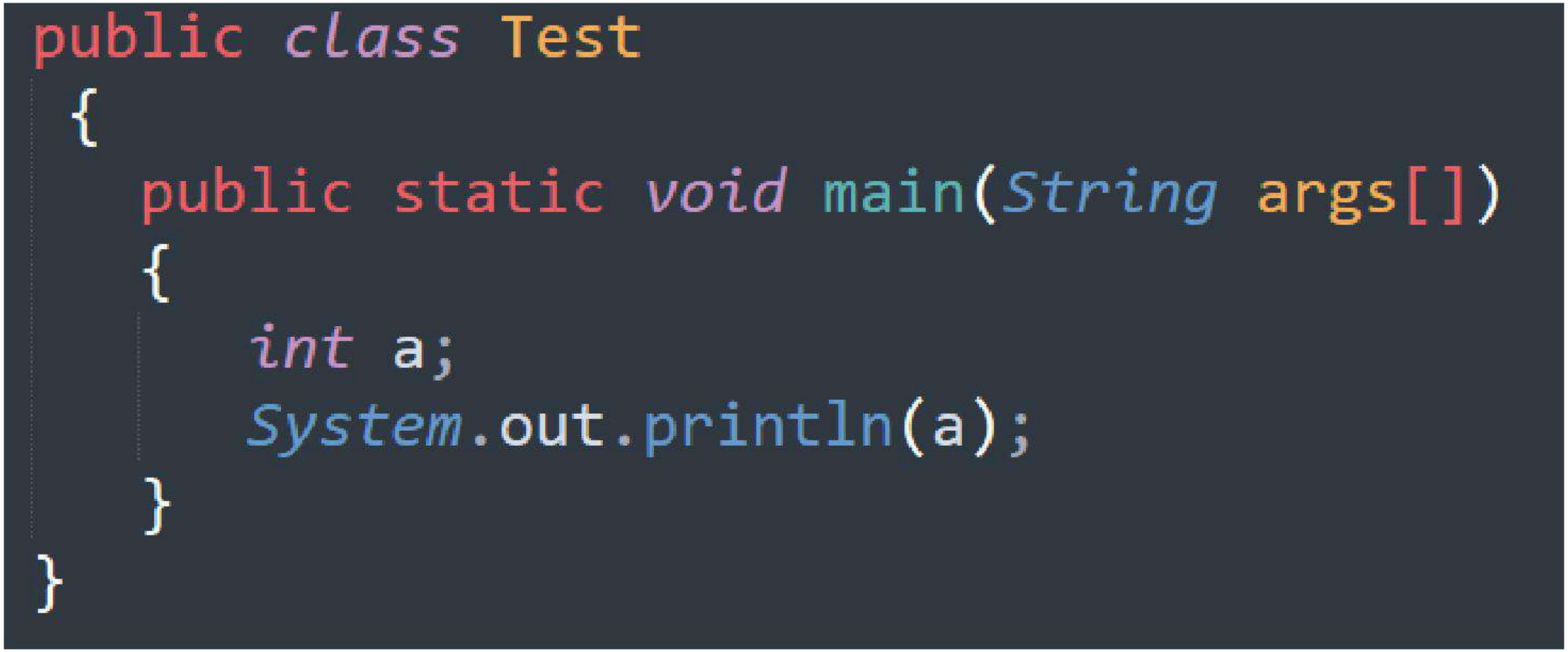


```
public class Test
 {
   public static void main(String args[])
   {
       int a;
       System.out.println(a);
   }
}
```

Types of Java constructors

Class Work Question

- The out of the following code is



```
public class Test
 {
   public static void main(String args[])
   {
      int a;
      System.out.println(a);
   }
}
```

- **error** because **the variable are not initialized**

Types of Java constructors

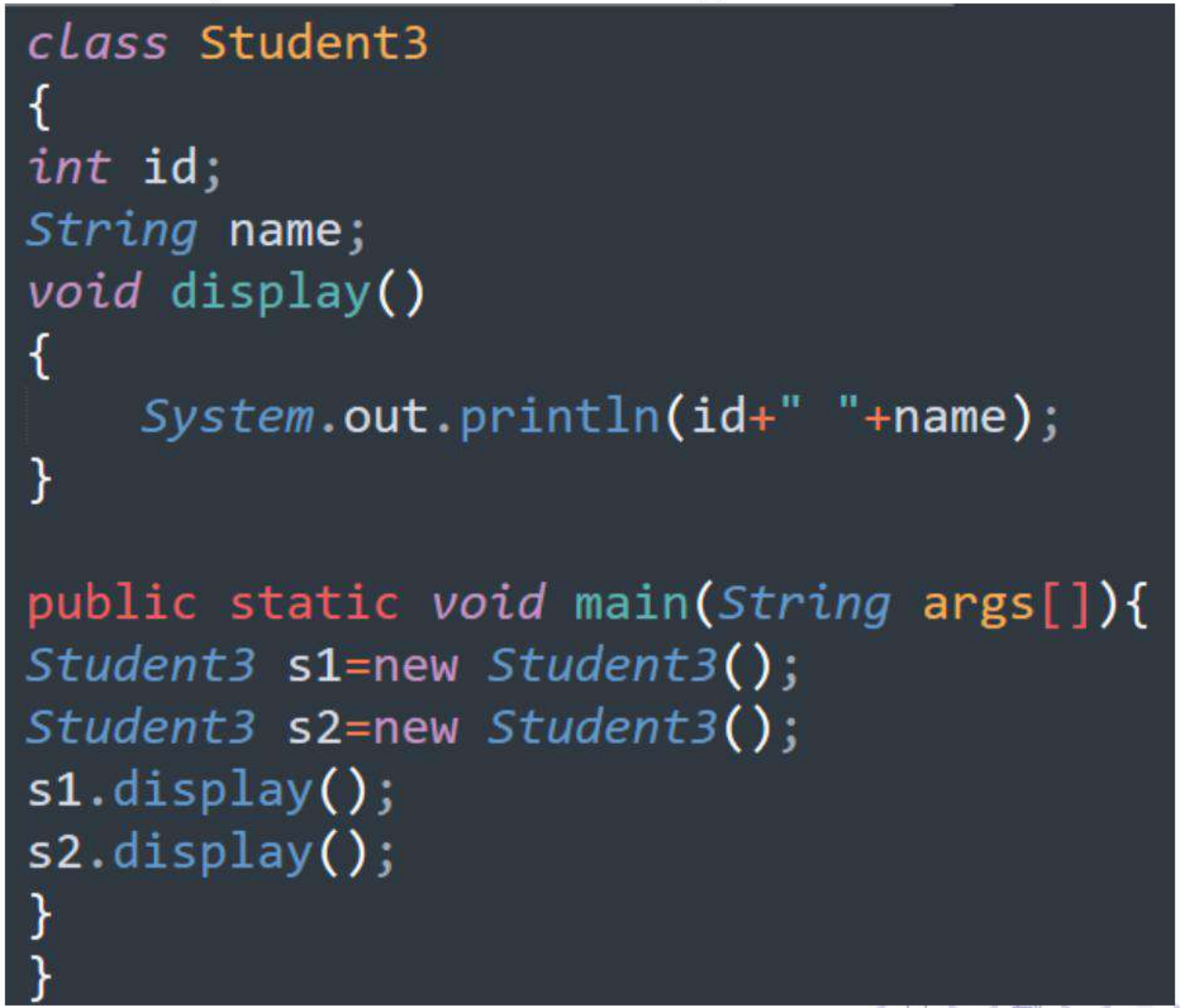
Class Work Question

- **What is the purpose of a default constructor?**
- **The answer is:**
  - **The purpose of the default constructor is to initialize the attributes of the object with their default values.**

Types of Java constructors

Example of default constructor

- **what is the output of the following code**



```
class Student3
{
int id;
String name;
void display()
{
    System.out.println(id+" "+name);
}

public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

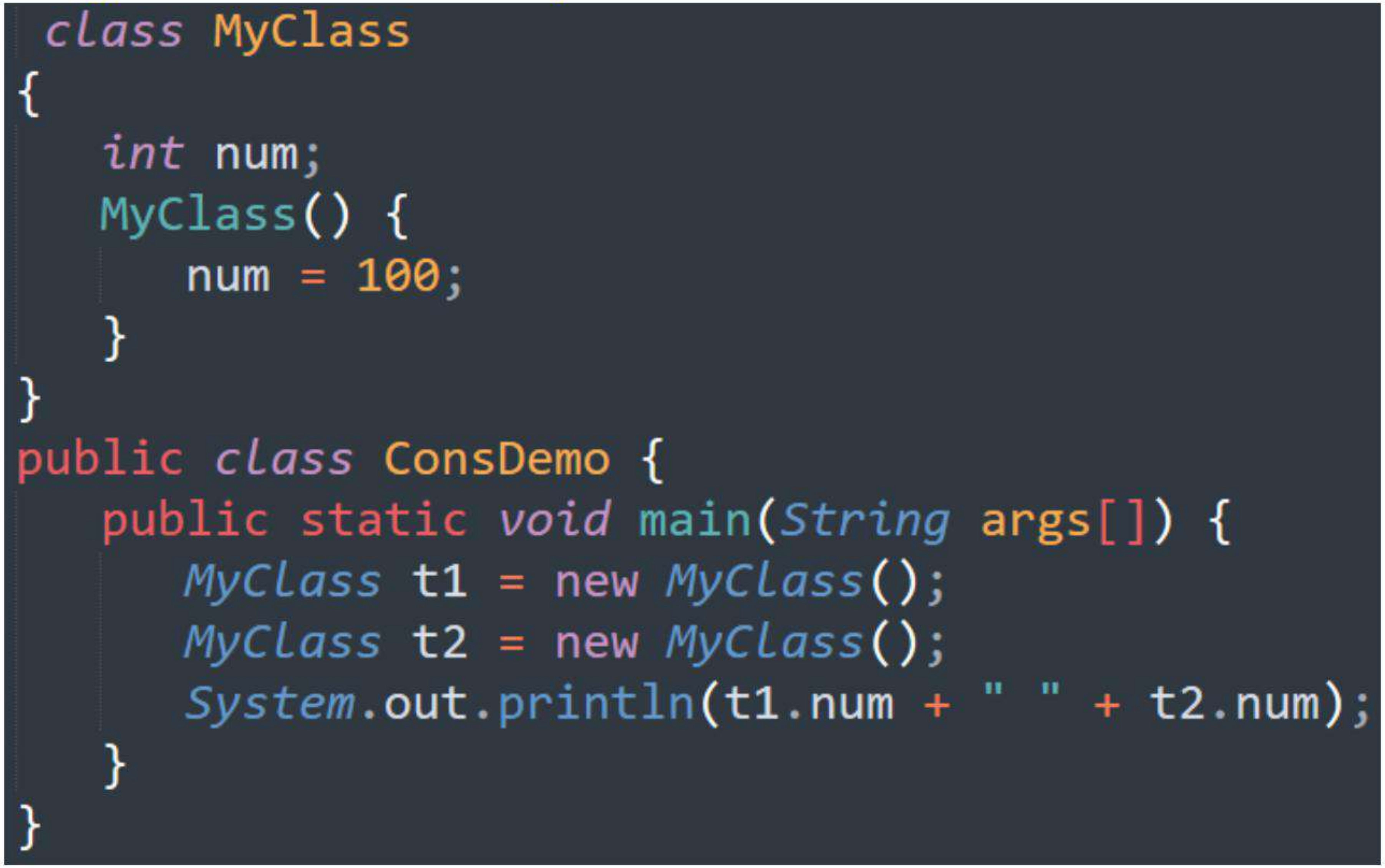
Types of Java constructors

No-Args Constructor

- Constructor **without** any argument is called a no-args constructor.
- The **signature is same as default constructor**
  - **However body can have any code** unlike default constructor where the **body of the constructor is empty.**

Types of Java constructors

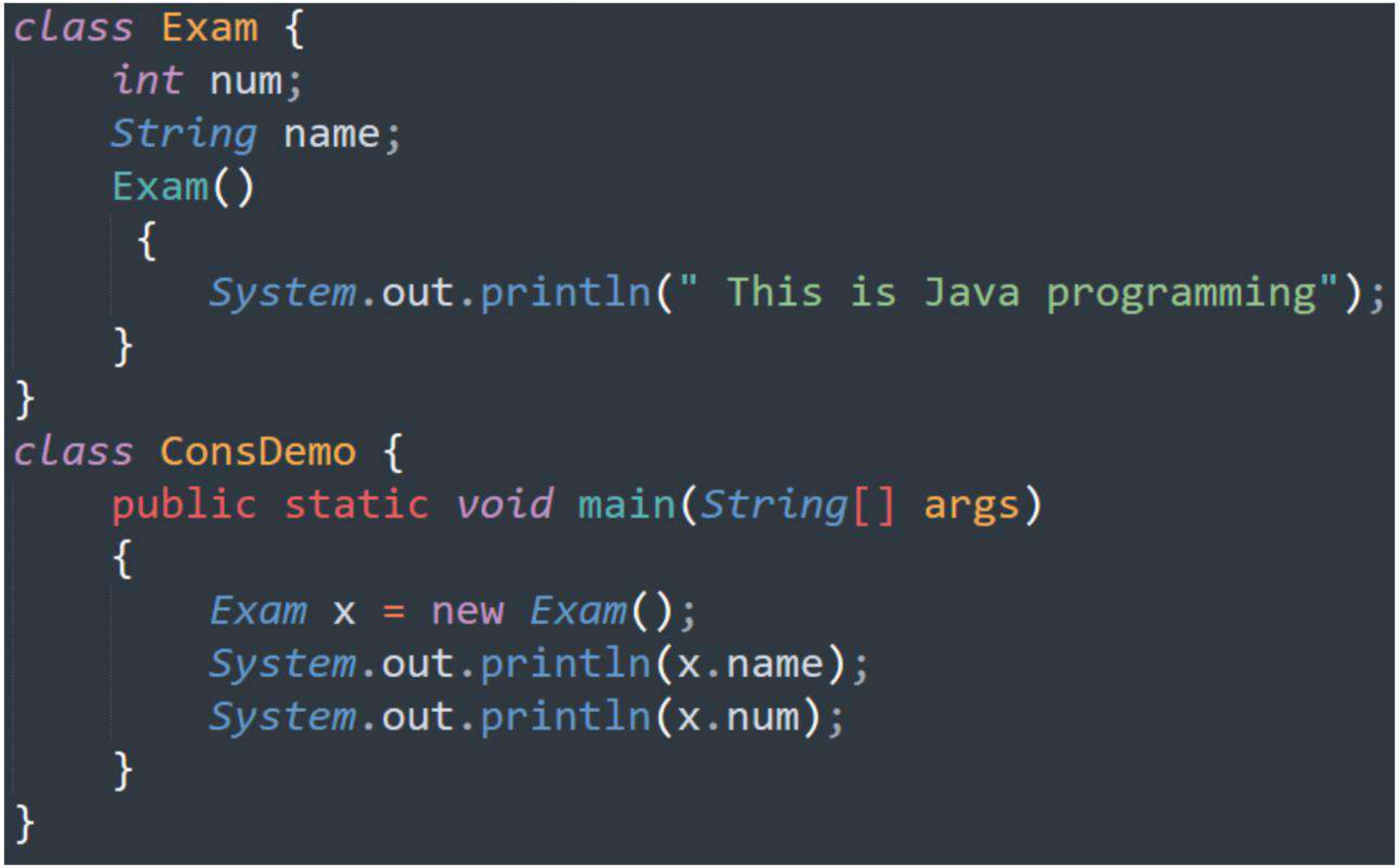
No-Args Constructor Examples



```
 class MyClass
{
   int num;
   MyClass() {
      num = 100;
   }
}
public class ConsDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.num + " " + t2.num);
   }
}
```

Types of Java constructors

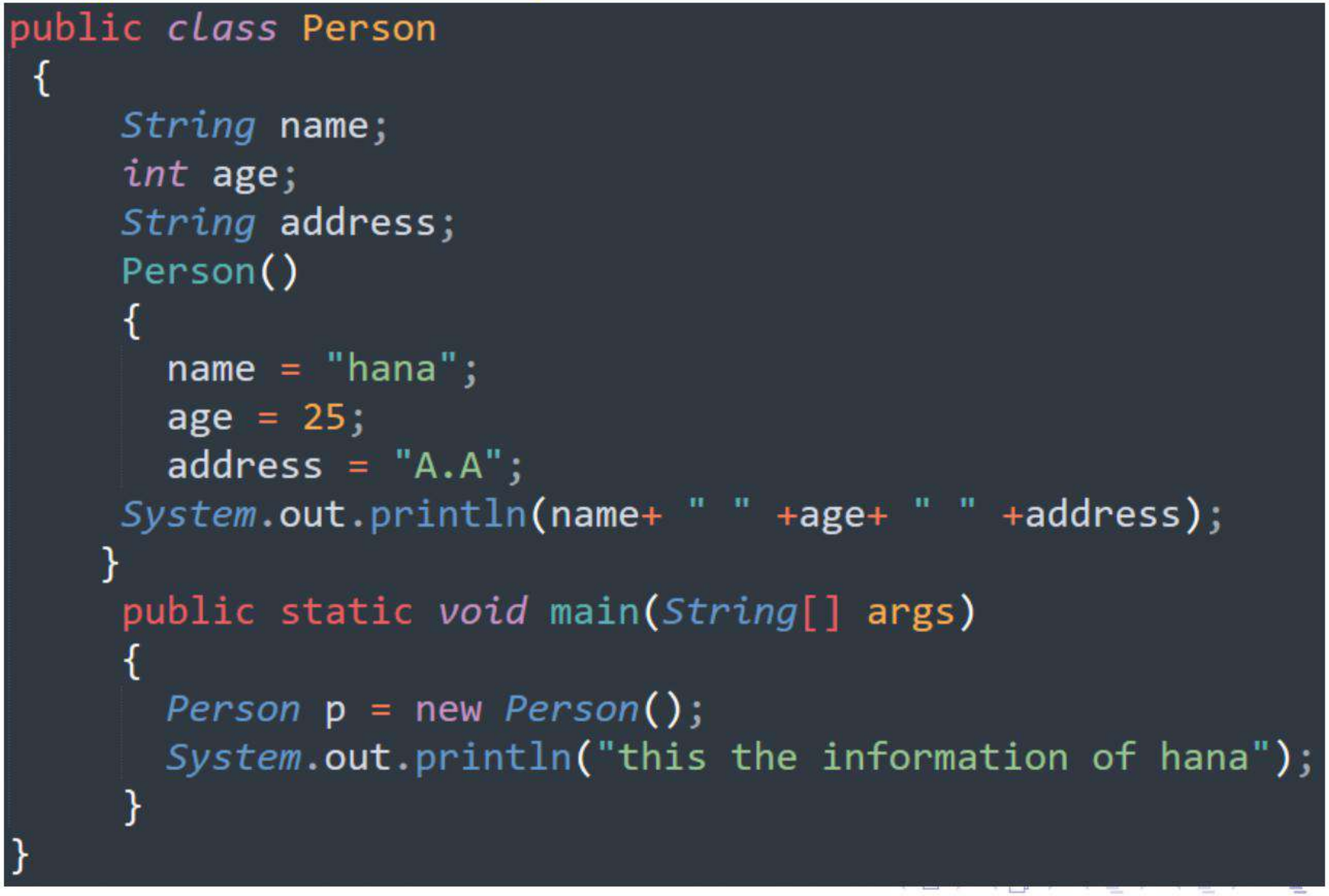
No-Args Constructor Examples



```
class Exam {
    int num;
    String name;
    Exam()
     {
        System.out.println(" This is Java programming");
    }
}
class ConsDemo {
    public static void main(String[] args)
    {
        Exam x = new Exam();
        System.out.println(x.name);
        System.out.println(x.num);
    }
}
```

Types of Java constructors

No-Args Constructor Examples



```
public class Person
 {
     String name;
     int age;
     String address;
     Person()
     {
       name = "hana";
       age = 25;
       address = "A.A";
     System.out.println(name+ " " +age+ " " +address);
    }
     public static void main(String[] args)
     {
       Person p = new Person();
       System.out.println("this the information of hana");
     }
}
```

Types of Java constructors

Parameterized Constructors

- A constructor which has **a specific number of parameters is called a parameterized constructor.**
- If we want to **initialize fields** of the class with our own values, then **use a parameterized constructor.**

Types of Java constructors

Parameterized Constructors

- **Why use the parameterized constructor?**

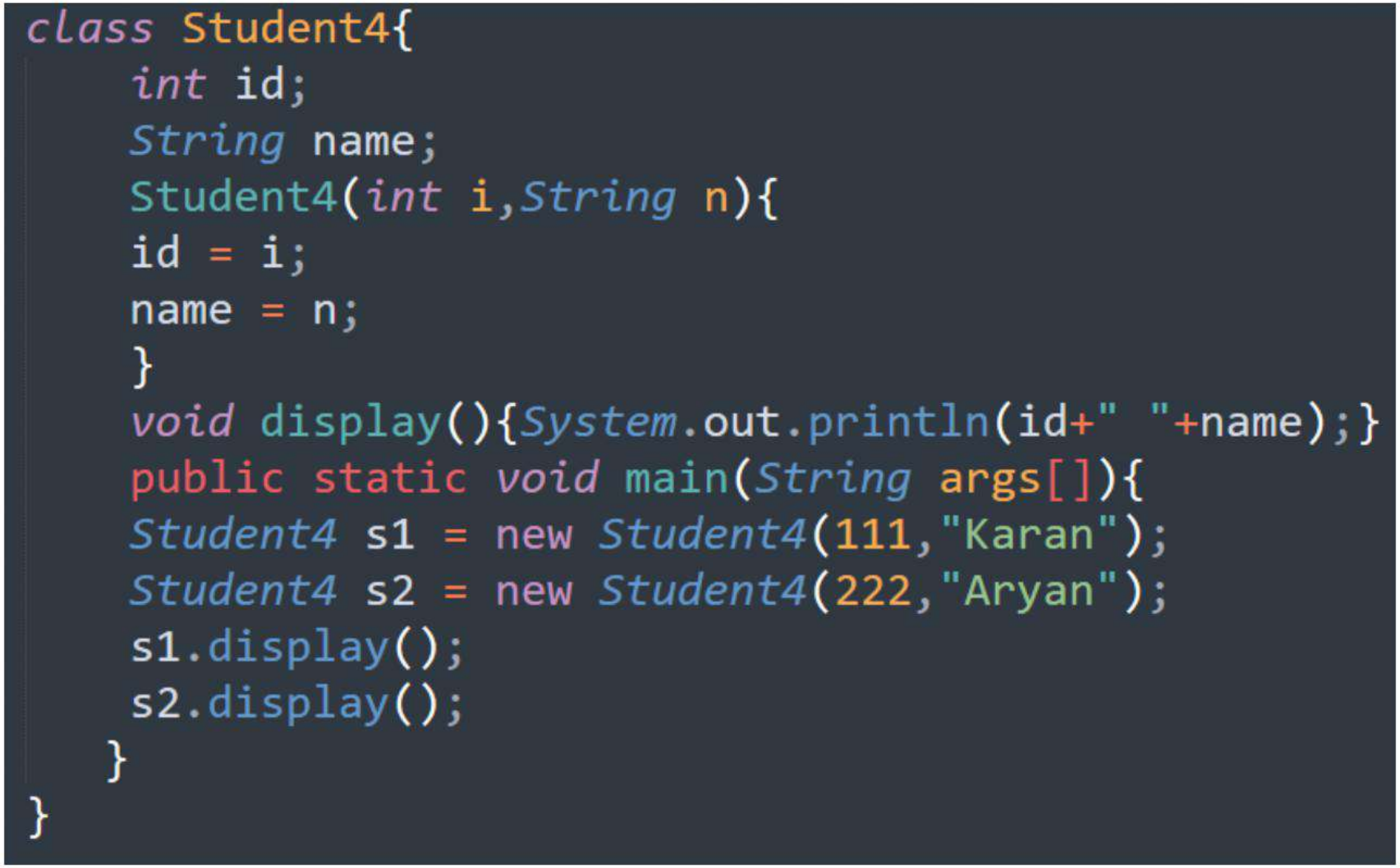
Types of Java constructors

Parameterized Constructors

- **Why use the parameterized constructor?**
- **The Answer is:**
  - The parameterized constructor **is used to provide different values to distinct objects.**
  - **However**, you can provide the **same values also.**

Types of Java constructors

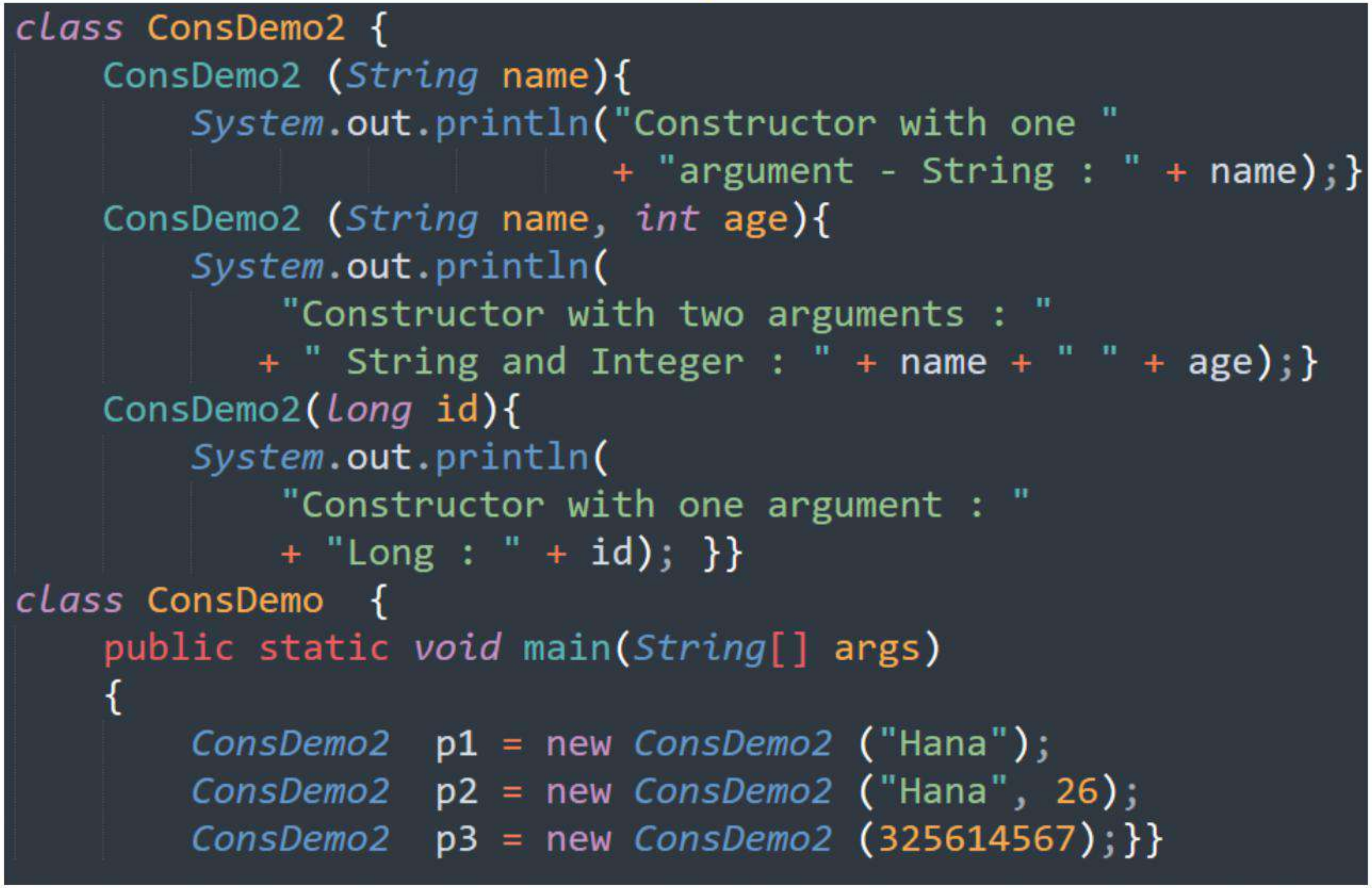
Parameterized Constructors Example



```java
class Student4{
    int id;
    String name;
    Student4(int i,String n){
    id = i;
    name = n;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Types of Java constructors

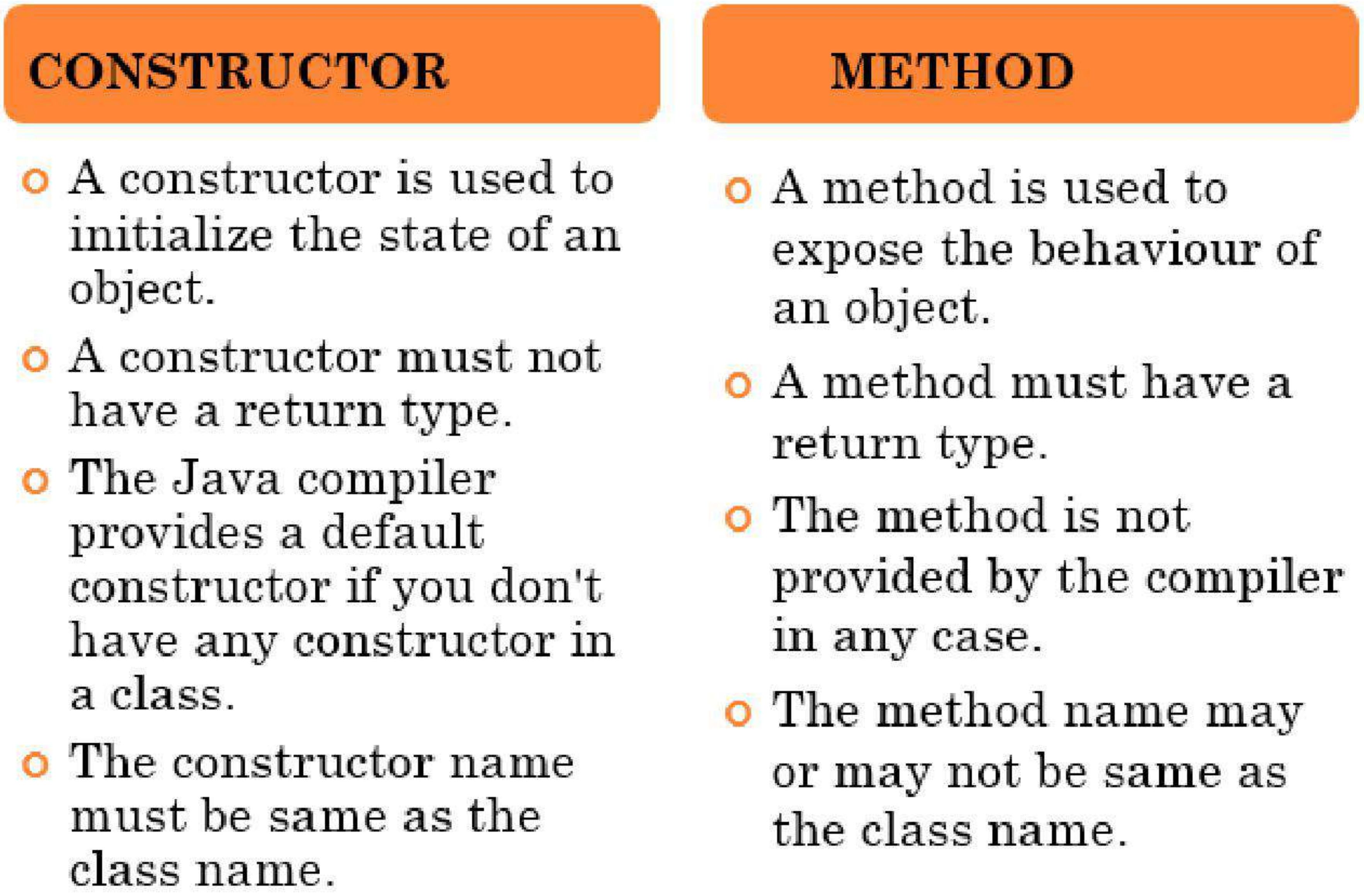
Parameterized Constructors Example



```
class ConsDemo2 {
    ConsDemo2 (String name){
        System.out.println("Constructor with one "
                        + "argument - String : " + name);}
    ConsDemo2 (String name, int age){
        System.out.println(
            "Constructor with two arguments : "
           + " String and Integer : " + name + " " + age);}
    ConsDemo2(long id){
        System.out.println(
            "Constructor with one argument : "
            + "Long : " + id); }}
class ConsDemo  {
    public static void main(String[] args)
    {
        ConsDemo2  p1 = new ConsDemo2 ("Hana");
        ConsDemo2  p2 = new ConsDemo2 ("Hana", 26);
        ConsDemo2  p3 = new ConsDemo2 (325614567);}}
```

Types of Java constructors

Difference between constructor and method in Java



CONSTRUCTOR	METHOD
A constructor is used to initialize the state of an object.	A method is used to expose the behaviour of an object.
A constructor must not have a return type.	A method must have a return type.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Types of Java constructors

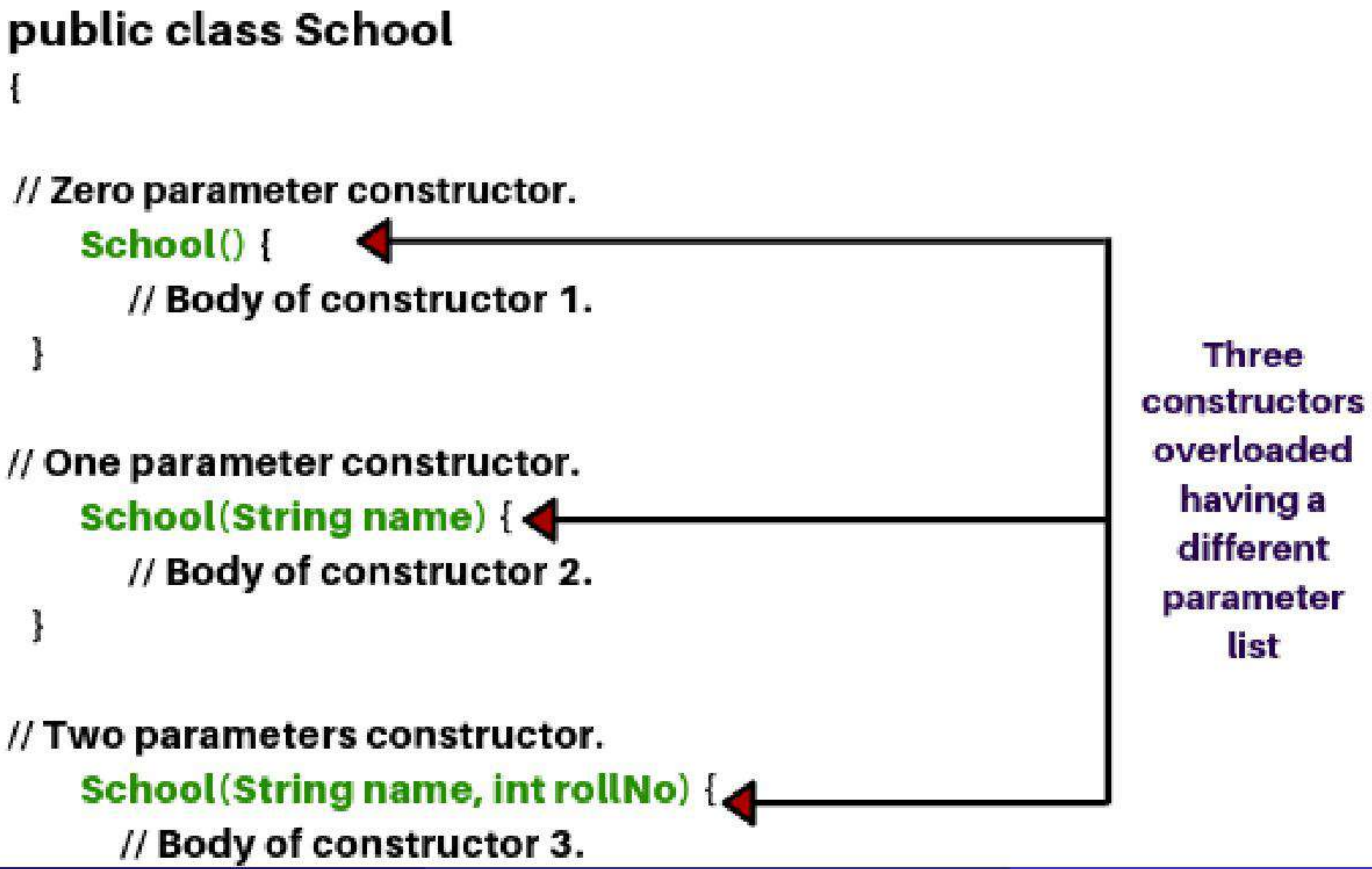
Constructor Overloading

- **Constructor overloading** is a concept of having **more than one constructor** with **different parameters list**
  - In such a way so that each **constructor performs a different task.**
- **Two or more constructors** with the **same name** but with **different signatures** is called **constructor overloading**.
- **If two constructors** of a class have the **same signature**, it represents **ambiguity**.
  - **In this case**, **Java compiler** will generate **an error message because** Java compiler will **unable to differentiate which form to execute**.

Types of Java constructors

Constructor Overloading

- **Java compiler** decides **which constructor** has to be called **depending** on **the number of arguments passing with objects.**

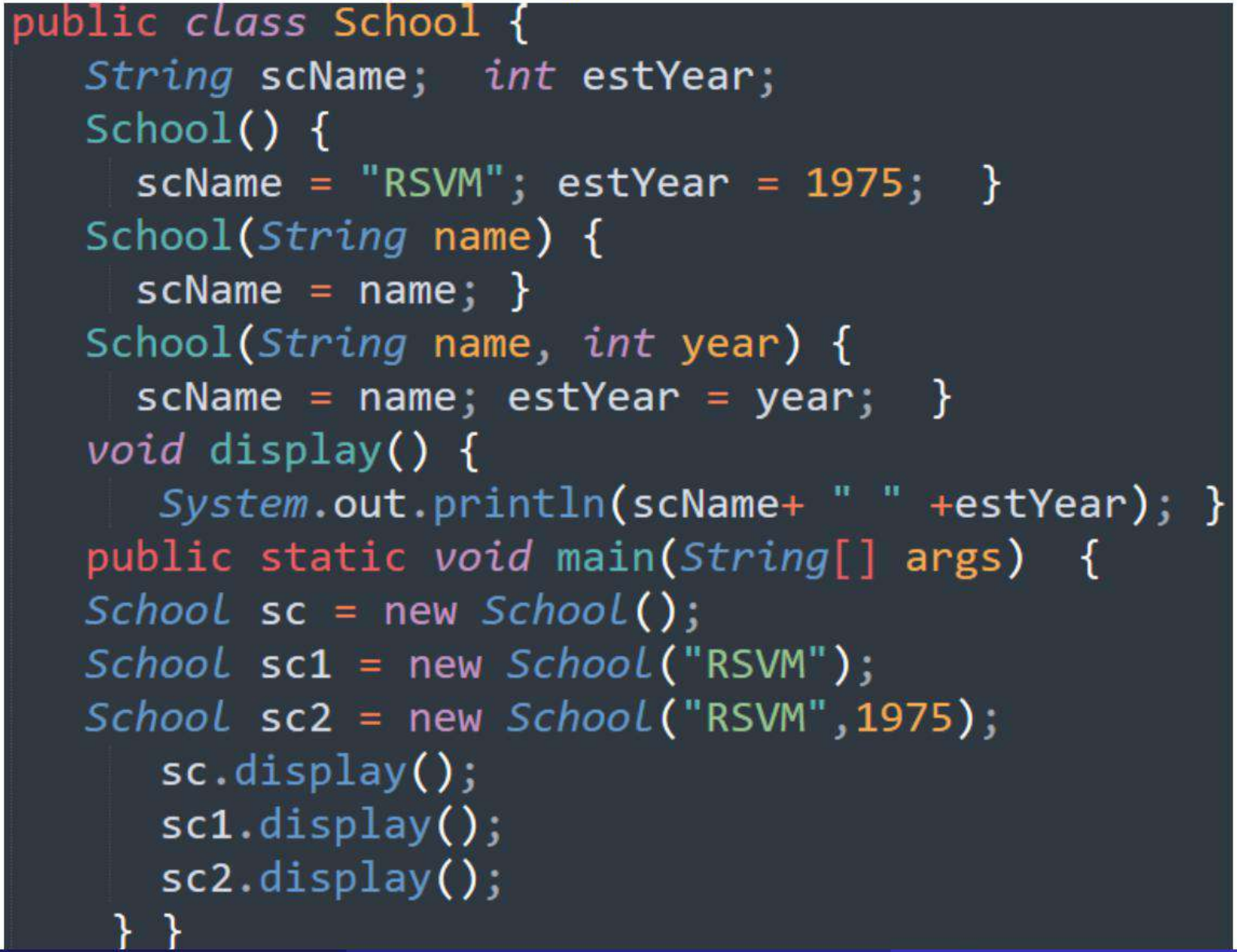


```
public class School
{

 // Zero parameter constructor.
     School() {
        // Body of constructor 1.
  }

// One parameter constructor.
     School(String name) {
        // Body of constructor 2.
  }

// Two parameters constructor.
     School(String name, int rollNo) {
        // Body of constructor 3.
```

Three constructors overloaded having a different parameter list

Types of Java constructors

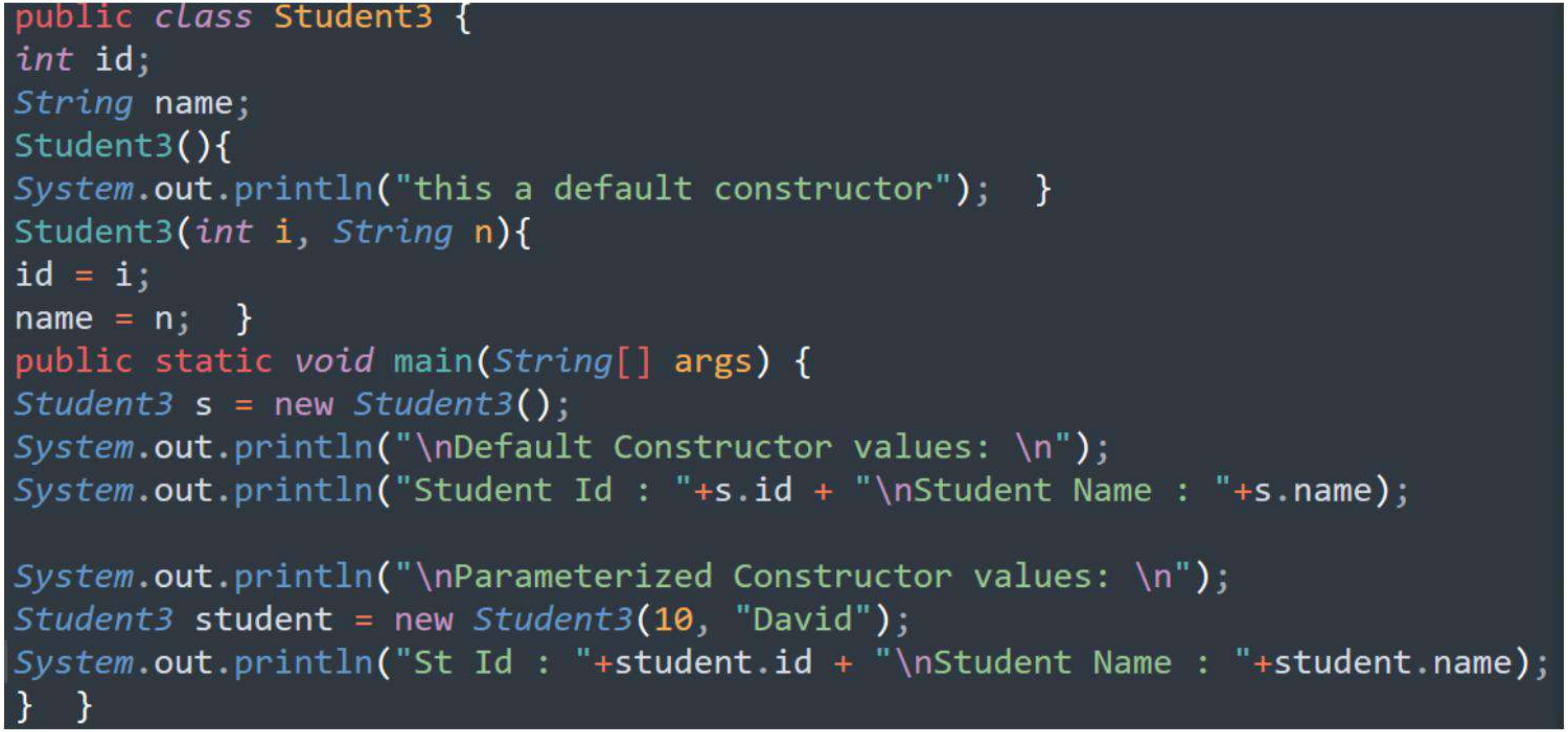
Constructor Overloading Example



```
public class School {
   String scName;  int estYear;
   School() {
     scName = "RSVM"; estYear = 1975;  }
   School(String name) {
     scName = name; }
   School(String name, int year) {
     scName = name; estYear = year;  }
   void display() {
      System.out.println(scName+ " " +estYear); }
   public static void main(String[] args)  {
   School sc = new School();
   School sc1 = new School("RSVM");
   School sc2 = new School("RSVM",1975);
      sc.display();
      sc1.display();
      sc2.display();
    } }
```

Types of Java constructors

Constructor Overloading Example



```
public class Student3 {
int id;
String name;
Student3(){
System.out.println("this a default constructor");  }
Student3(int i, String n){
id = i;
name = n;   }
public static void main(String[] args) {
Student3 s = new Student3();
System.out.println("\nDefault Constructor values: \n");
System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);

System.out.println("\nParameterized Constructor values: \n");
Student3 student = new Student3(10, "David");
System.out.println("St Id : "+student.id + "\nStudent Name : "+student.name);
}  }
```

●

End of Chapter Four

- Thank You!!!