Sri Krishna Arts and Science College

Java Programming

Facilitator:

S.Vetrivel

Assistant Professor

Department of ICT & Cognitive Systems

SRI KRISHNA INSTITUTIONS COIMBATORE

Sri Krishna Arts and Science College

CLO	Statements	Level of Taxonomy / Mode of Assessment
CLO1	Explain the concepts of Object-Oriented Programming and the basics of java language.	C2 (Cognitive) Understand **1. Quiz**
CLO2	Elucidate the concepts of java classes, objects, inheritance, enumeration, packages, exception- handling mechanism, Assertions, Serialization and Cloning.	C4 (Cognitive) CIA & ESE
CLO3	Practice OOPs concepts and advanced features.	A2 (Affective) **2. Group Assignments** **3. Group Reports**
CLO4	Apply the concepts of Applets, Event handling, Swings and JDBC components.	C3 (Cognitive) CIA & ESE
CLO5	Propose a GUI based java application using Applets, or swings with JDBC component.	A4 (Affective) **4. Assignment** **5.Project Report**

JAVA PROGRAMMING

C++	Java
Bjarne Stroustrup at Bell Labs in 1979	James Gosling at Sun Microsystems.
Support for both procedural programming and object-oriented programming.	support only for object-oriented programming models
Supports Header Files	Only Packages
Platform dependent	Platform Independent
Supports features like operator overloading, Goto statements, structures, pointers, unions, etc.	Does not support features like operator overloading, Goto statements, structures, pointers, unions, etc.
Only Compiled and cannot be Interpreted.	Both compiled and interpreted.
Limited libraries	More diverse libraries.
Calls to native system libraries.	Java Native Access are allowed
Memory management is manual.	Memory management is System controlled.

JAVA PROGRAMMING

C++	Java
Both global and namespace scopes are supported.	No support for global scope
C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Does not support unsigned right shift	Support unsigned right shift
Game engines, Machine learning, Operating systems, Google Search Engine, Web browsers, Virtual Reality (VR), Game development, Medical technology, Telecommunications, Databases, etc.	Internet and Android games, Mobile applications, Healthcare and research computation, Cloud applications, Internet of Things (IoT) devices, etc

JAVA PROGRAMMING

Introduction to Java:

Object-oriented programming has several advantages over procedural programming:

- ❖OOP is faster and easier to execute
- ❖OOP provides a clear structure for the programs
- ❖OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- ❖OOP makes it possible to create full reusable applications with less code and shorter development time

History of Java

- Java is an object oriented, general purpose, multithreaded programming language developed by Sun Microsystems of USA in 1991.
- Java was conceived by **James Gosling, Patrick Naughton, Chris Warth, and Ed Frank.**
- This Language was originally called "**Oak**" but was renamed as "**Java**" in 1995.
- The current version of Java is **Java 20.**
- The current long-term support version (LTS) of Java is **Java 17**, released in September 2021

Getting Started with java

- ❖ Java is a general purpose, object oriented language.

- ❖ We can develop two types of java programs.

1. Stand alone applications

2. Web application (applets)

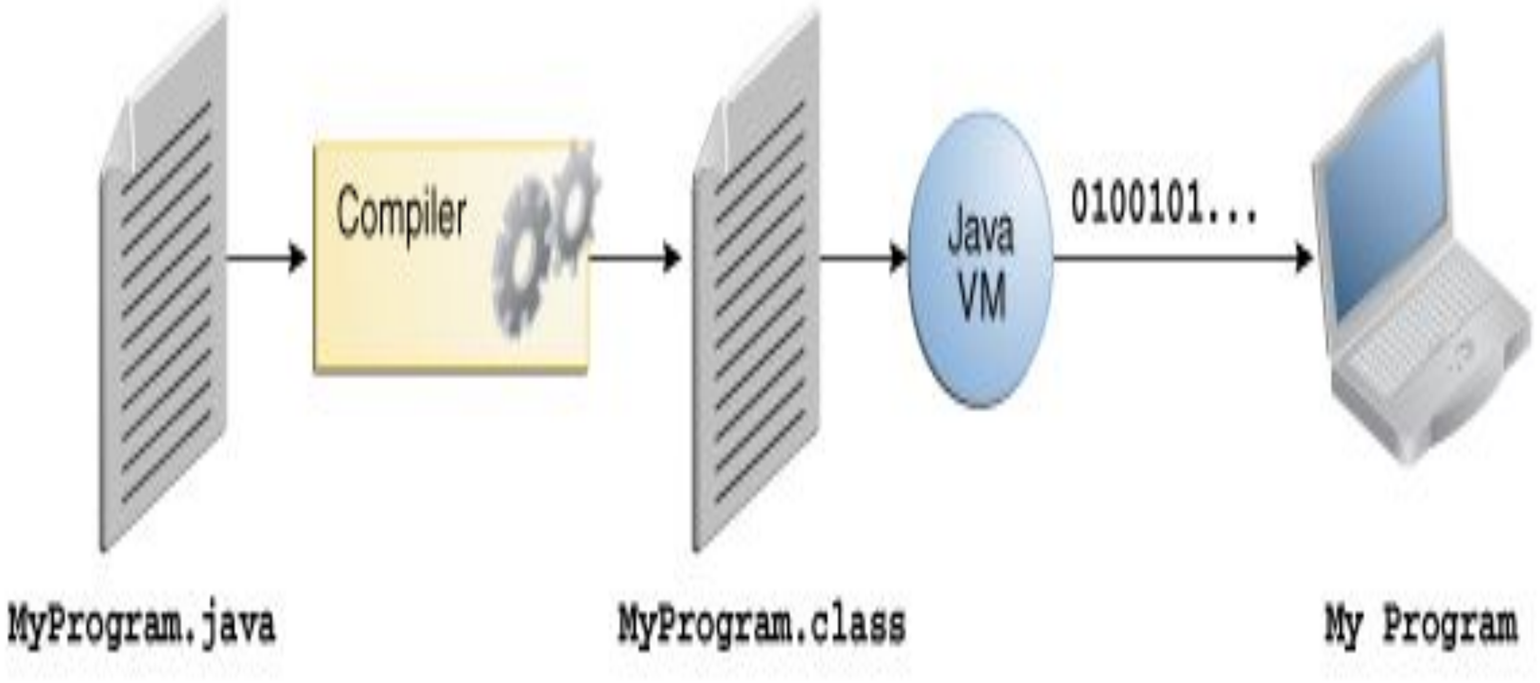
- ❖ Stand alone applications are programs written in java to carry out certain tasks on a stand-alone local computer

Getting Started with java

- Executing a java stand-alone program involves two steps
- Compiling source code into byte code using javac complier
- Executing the byte code program using java interpreter.
- Applets are small java program developed for internet applications

Getting Started with java

- An applet located on a distant computer (server) can be downloaded via internet and executed on a local computer (client) using Java-capable browser.
- Stand-alone programs can read and write files and perform certain operations that applets cannot do.
- An applet only can run within a web browser


MyProgram.java
Compiler
MyProgram.class
Java VM
0100101...
My Program

Java Virtual Machine

- ✔ The Java Virtual Machine is called JVM, is an abstract computing machine or virtual machine interface that drives the java code.
- ✔ JVM is the engine that drives the Java code.
- ✔ Mostly in other Programming Languages, compiler produce code for a particular system but

  Java compiler produce Bytecode for a Java Virtual Machine

Java Virtual Machine

- ✔ JVM Works in the following Manner
- ✔ Reading Bytecode.
- ✔ Verifying bytecode.
- ✔ Linking the code with the library.
- ✔ JVM generates a .class(Bytecode) file, and that file can be run in any OS, but JVM should present in OS because JVM is platform dependent.
- ✔ Java applications are called WORA (Write Once Run Anywhere). This is because of JVM.

Java Virtual Machine

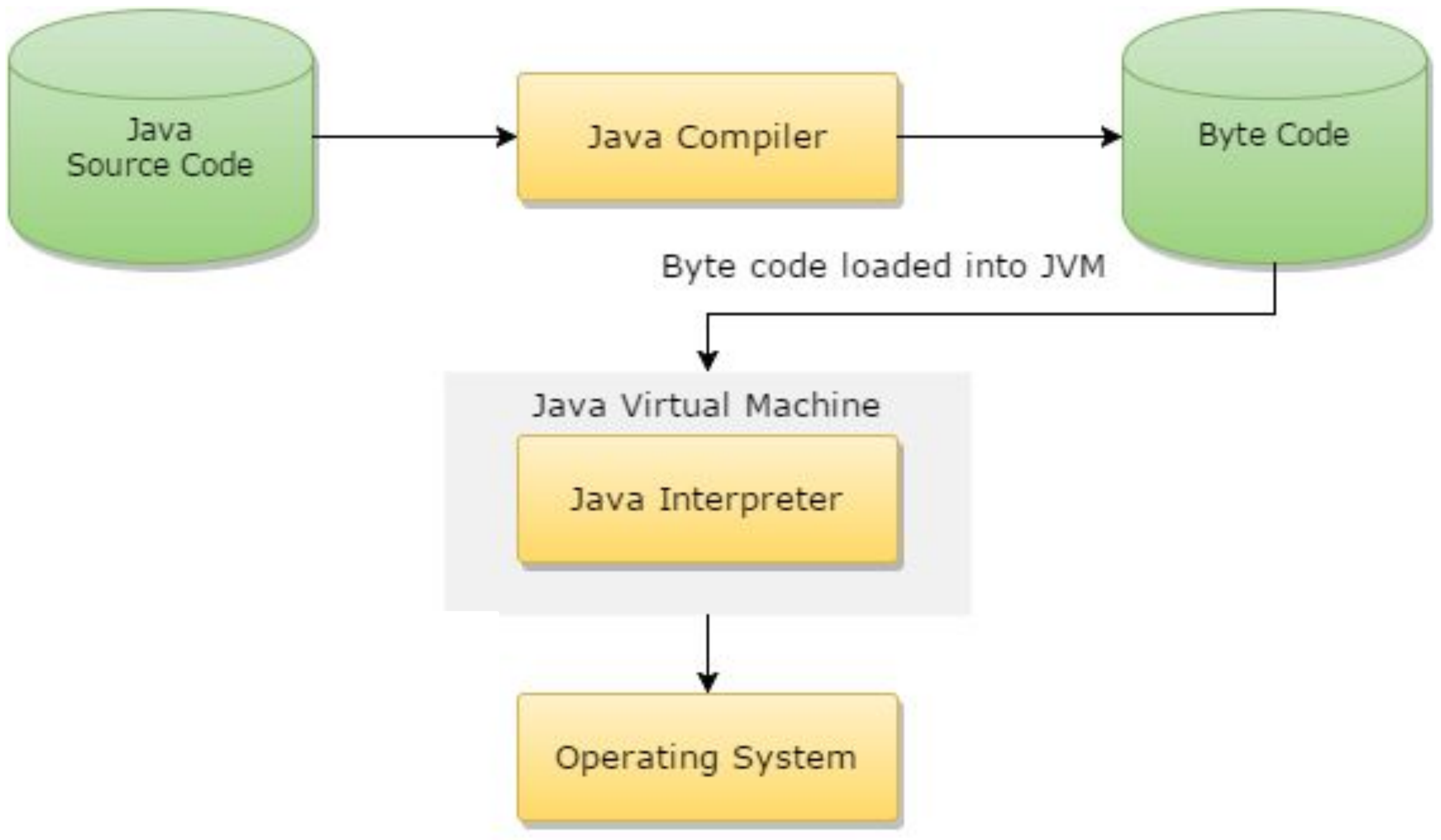
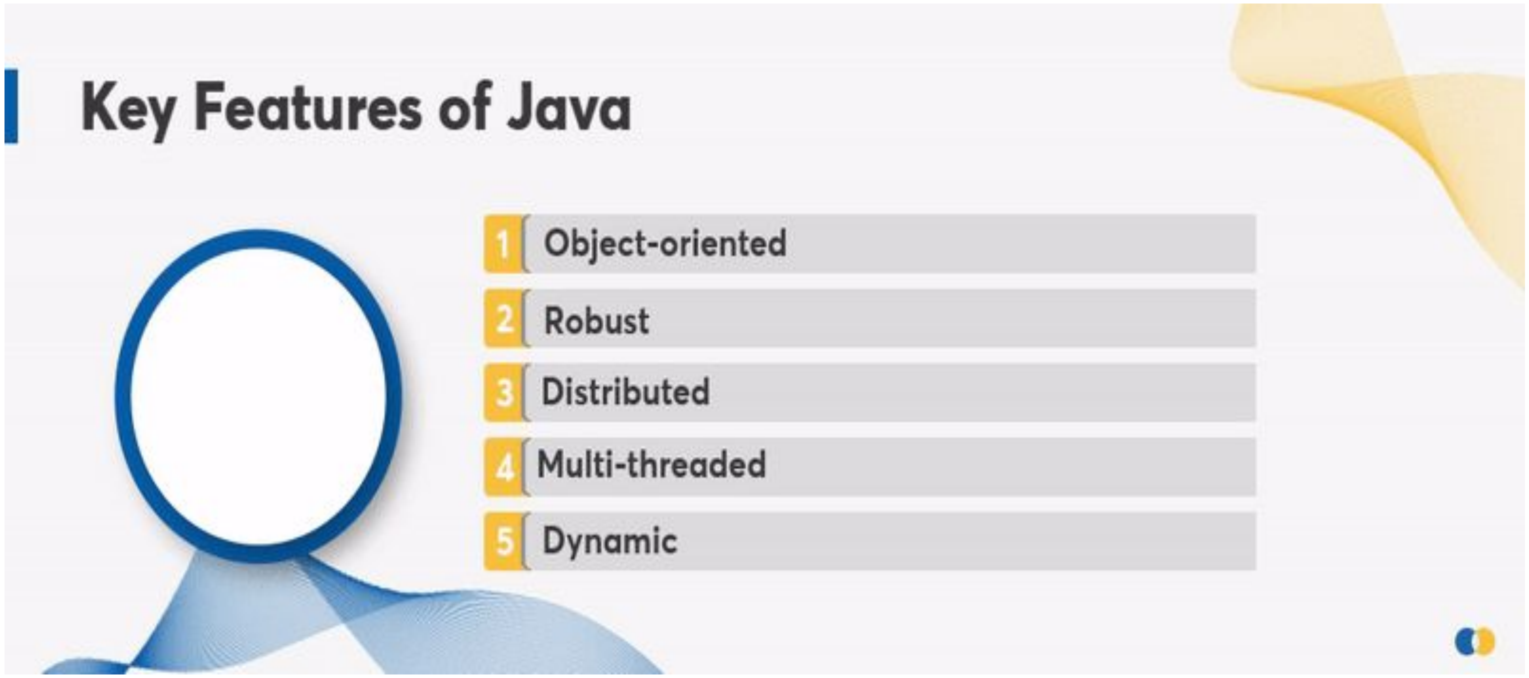


Diagram of JVM



Key Features of Java

1. Object-oriented
2. Robust
3. Distributed
4. Multi-threaded
5. Dynamic

- ✔ Simple
- ✔ Portable
- ✔ Secure

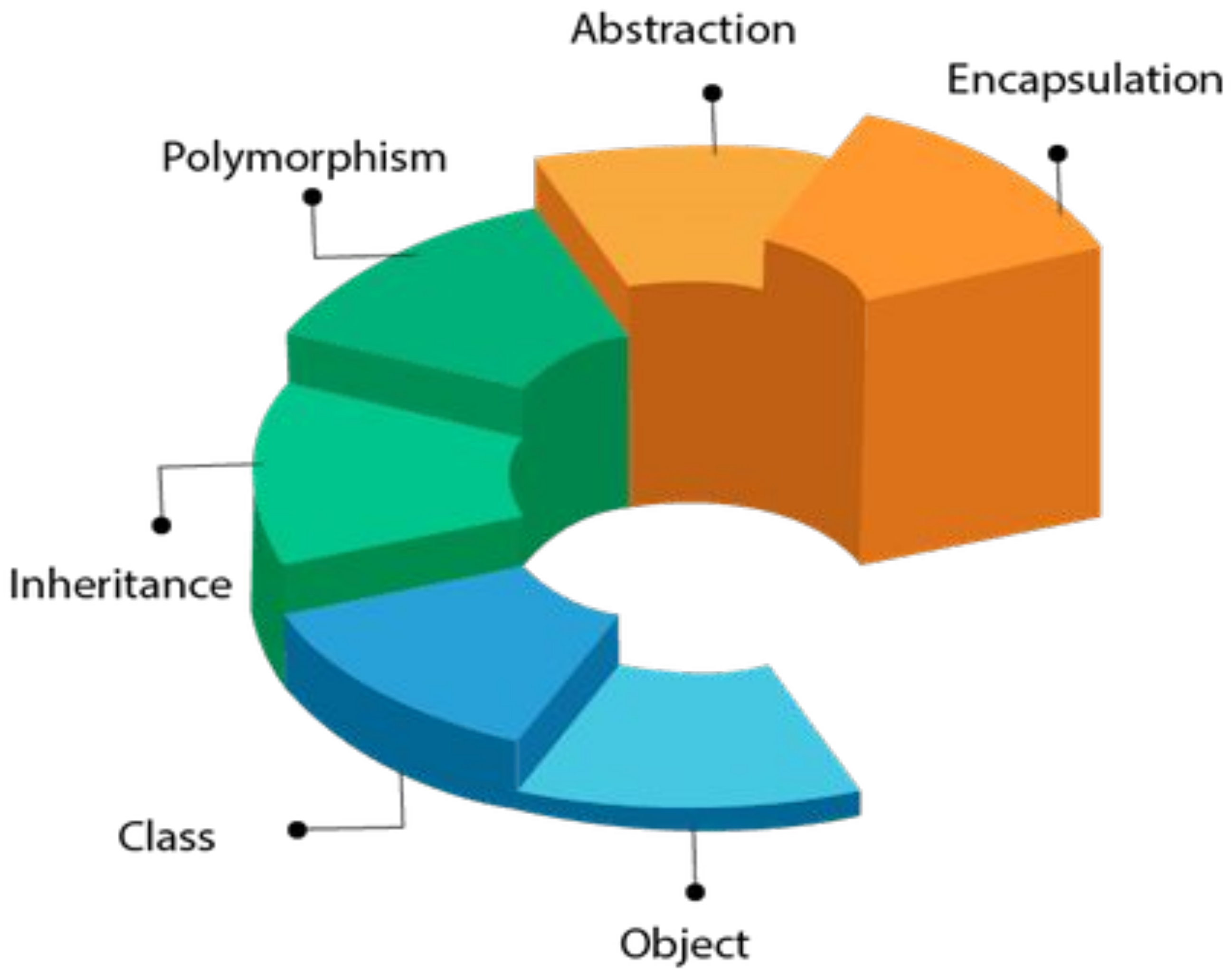
Architecture neutral

High performance

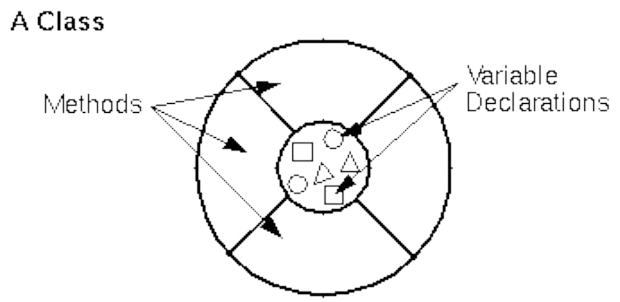
CHARACTERISTICS OF JAVA

- Platform independent
- Object-orientated programming language
- Strongly-typed programming language
- Interpreted and compiled language
- Automatic memory management

OOPs (Object-Oriented Programming System)



OOP's concept



Test Yourself….

❖ Which of the following option leads to the portability and security of Java?

a) Bytecode is executed by JVM

b) The applet makes the Java code secure and portable

c) Use of exception handling

d) Dynamic binding between objects

Test Yourself….

❖ Which of the following is not a Java features?

a) Dynamic

b) Architecture Neutral

c) Use of pointers

d) Object-oriented

Test Yourself….

❖ ______ is used to find and fix bugs in the Java programs.

a) JVM

b) JRE

c) JDK

d) JDB

Test Yourself….

❖ Which of the following is an immediate subclass of the Panel class?

a) Applet class

b) Window class

c) Frame class

d) Dialog class

Thank you

Sri Krishna Arts and Science College

Programming In Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

Program Structure

- A Java application consists of a collection of classes.
- A class is a template.
- An object is defined as an instance of the class.
- Each instance (object) contains the members (fields and methods) specified in the class.
- A field is one that holds a value.
- A method defines operations on the fields and values that are passed as arguments to the method

First Java Program

```
 Call this file"Example.java".
class Example
 {
      //your program starts execution with a call to main()
public static void main (String args[ ])
    {
      System.out.println(“This is a simple Java program”);
    }
}
```

How to Execute a Java Program?

- There are three easy steps for successfully executing the Java program:
- 1. Entering the Source Code
- 2. Saving the Source Code
- 3. Compiling and Running the Source
- javac Example.java
- java Example

Java Programming Constructs

VARIABLES

- Variable is a symbolic name refer to a memory location used to store values that can change during the execution of a program.
- Java declares its variables in the following manner:
- Eg: int ant = 100;
- A variable declaration involves specifying the type (data type), name (identifier), and value (literal) according to the type of the variable

PRIMITIVE DATA TYPES

- Primitive data types are the basic building blocks of any programming language.
- A primitive data type can have only one value at a time and is the simplest built-in form of data within Java.
- There are eight primitive data types in Java, as follows:
- For whole number ----byte short int long
- For real numbers---- float double
- Characters--- char
- Boolean---- boolean

IDENTIFIER

- Identifiers are names assigned to variables, constants, methods, classes, packages, and interfaces.
- No limit has been specified for the length of a variable name. Identifiers can have letters, numbers, underscores, and any currency symbol.
- However they may only begin with a letter, underscore, or a dollar sign.
- Digits cannot be the first character in an identifier.

IDENTIFIER

- **<u>Rules for Naming</u>**
- 1. The fi rst character of an identifi er must be a letter, an underscore, or a dollar sign ($).
- 2. The subsequent characters can be a letter, an underscore, dollar sign, or a digit.
- Note that white spaces are not allowed within identifiers.
- 3. Identifiers are case-sensitive.
- Do not use Java's reserved keywords.
- A few examples of legal and illegal identifiers are shown below.

LITERALS

- A literal is a value that can be passed to a variable or constant in a program.
- Literals can be numeric (for byte, short, int, long, float, double), boolean, character, string notations or null literals.
- Numeric Literals can be represented in binary, decimal, octal, or hexadecimal notations

LITERALS

- Binary literals are a combination of 0’s and 1’s. Binary literals can be assigned to variables in Java 7.
- Binary literals must be prefixed with 0b or 0B (zerob or zeroB).
- octal literals have to be specified, the value must be prefixed with a zero and only digits from 0 to 7 are allowed.
- For example, int x = 011;

LITERALS

- Hexadecimal literals are prefixed with 0x or 0X; the digits 0 through 9 and a through f (or A through F) are only allowed.
- For example, int y = 0x0001;
- All integer literals are of type int, by default.
- To define them as long, we can place a suffix of L or l after the number for instance:
- long l = 2345678998L;

LITERALS

- All floating literals are of type double, by default.
- For instance, float f = 23.6F; double d = 23.6;
- For char literals, a single character is enclosed in single quotes.
- char c = '\u004E';
- A boolean literal is specified as either true or false.
- By default, it takes the value false

LITERALS

- String literals consist of zero or more characters within double quotes.
- For instance, String s = "This is a String Literal";
- Null literals are assigned to object reference variables (see Chapter 4 for object references).
- s = null;

謝謝你
Grazie
Gracias
Merci
Danke
Thanks!
THANK
YOU!

SRI KRISHNA
INSTITUTIONS
COIMBATORE

Sri Krishna Arts and Science College

Programming In Java

Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

OPERATORS

- An operator performs an action on one or more operands.
- An operator that performs an action on one operand is called a unary operator (+, –, ++, – –).
- An operator that performs an action on two operands is called a binary operator (+, –, / , * , and more).
- An operator that performs an action on three operands is called a ternary operator (? :).
- Java provides all the three operators.

OPERATORS

- Binary Operators Java provides arithmetic, assignment, relational, shift, conditional, bitwise, and member access operators.

Assignment Operators:

- The simple ‘=’ operator sets the left-hand operand to the value of the right-hand operand.
- Java supports the following list of shortcut or compound assignment operators:
- += –= *= /= %= &= |= ^= <<=>>= >>>=

OPERATORS

Arithmetic Operators:

- Arithmetic operators are used for adding (+), subtracting (–), multiplying (*), dividing (/), and finding the remainder (%).
- Java does not support operator overloading
- compound assignment operators are also provided by Java.

OPERATORS

a += b; // evaluated as a = a + b;

a –= b; // evaluated as a = a – b;

a *= b; // evaluated as a = a * b;

a /= b; // evaluated as a = a / b;

a % = b; // evaluated as a = a % b

OPERATORS

```
class ArithmeticDemo
{
    public static void main(String args[])
    {
      int a = 25, b = 10;
     System.out.println("Sum "+ a +" +" + b +" = " + (a + b));
     //adding two variables a and b
     System.out.println("Subtraction "+ a +" - " + b +" = " + (a - b));
      //multiplying a with b
     System.out.println("Multiplication "+ a +" * " + b +" = " + (a * b));
```

OPERATORS

```
// Division System.out.println("Division "+ a +" / " + b +" = " + (a / b));

 // Remainder Operator

 System.out.println("Remainder "+ a +" % " + b +" = " + (a % b));

 // a and b can be added and the result can be placed in a // Let us see
    how?

a += b;

System.out.println("Added b to a and stored the result in a " + a);

 } }
```

OPERATORS

Relational Operators

- Relational operators in Java return either true or false as a boolean type.
- Relational Operators
- equal to ==
- Not equal to !=
- less than <
- greater than >
- less than or equal to <=
- greater than or equal to >=

OPERATORS

```
class RelationalOperatorDemo {
public static void main(String args[]) {
 int a = 10,b = 20;
 System.out.println("Equality Operator: a == b : \t\t\t" +(a == b));
 System.out.println("Not Equal To Operator: a != b : \t\t" +(a != b));
System.out.println("Less Than Operator: a == b : \t\t\t" +(a < b));
 System.out.println("Greater Than Operator: a == b : \t\t" +(a > b));
System.out.println("Less than or equal to Operator: a == b : \t" +(a <= b));
System.out.println("Greater than or equal to Operator: a == b : \t" +(a >= b));
    } }
```

OPERATORS

Boolean Logical Operators:

- Boolean logical operators are:
- conditional OR (||),
- conditional AND (&&),
- logical OR (|),
- logical AND (&),
- logical XOR (^),
- unary logical NOT (!).

OPERATORS

Bitwise Operators

- Bitwise operators include and, or, xor, not, right shift, left shift, and unsigned right shift.
- Bitwise shortcut operators include the following:
- AND: &=
- OR: |=
- XOR: ^=
- Shift Operator: >>=,<<=,>>>=

OPERATORS

```
class BitwiseOperatorDemo {

public static void main(String args[]) {

int x = 2,y = 3;

System.out.println("Bitwise AND: " +x+ "&" +y+ " = " +(x&y));

System.out.println("Bitwise OR : " +x+ " | " +y+ " = " +(x|y));

System.out.println("Bitwise XOR: " +x+ " ^ " +y+ " = " +(x^y));

System.out.println("Bitwise NOT: ~" +x+ " = " +(~x)); } }
```

OPERATORS

Unary Operators

- Unary operators, as the name suggest, are applied to only one operand.
- They are as follows: ++, - -, !, and ~.
- The unary boolean logical not (!) and bitwise logical not (~) have already been discussed.

OPERATORS

Increment and Decrement Operators

- Increment and decrement operators can be applied to all integers and floating-point types.
- They can be used either in prefix (– –x, ++x) or postfix (x– –, x++) mode.
- Prefix Increment/Decrement Operation
- int x = 2;
- int y = ++x; // x = 3, y = 3
- int z = --x; // x = 1, z = 1

OPERATORS

Postfix Increment/Decrement Operation

```
 int x = 2;

int y = x++; // x == 3, y == 2

int z = x--; // x = 1, z = 2
```

Ternary Operators

- Ternary operators are applied to three operands. This conditional operator (? :)
- operand1 ? operand2 : operand3

谢谢你
Grazie
Gracias
Merci
Danke
Thanks!
THANK
YOU!

SRI KRISHNA INSTITUTIONS COIMBATORE

Sri Krishna Arts and Science College

Programming In Java

Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

EXPRESSIONS

- An expression is a combination of operators and/or operands.
- Expressions may contain identifiers, types, literals, variables, separators, and operators.
- For example, int m = 2,n = 3,o = 4; int y = m * n * o;

PRECEDENCE RULES AND ASSOCIATIVITY

- Precedence rules are used to determine the order of evaluation priority in case there are two operators with different precedence.

EXPRESSIONS

- Associativity rules are used to determine the order of evaluation if the precedence of operators is same.
- Associativity is of two types: Left and Right.
- Precedence and associativity can be overridden with the help of parentheses.

EXPRESSIONS

```
class AssociativityAndPrecedenceTest
{
    public static void main(String[] args)
     {
        //precedence of * is more than that of + L1
        System.out.println(" 2 + 3 * 2 = \t " + (2 + 3 * 2));
        System.out.println(" 2 * 5 / 3 = \t " + (2 * 5 / 3 ));
        System.out.println("(2 + 3) * 2 = \t " + ((2 + 3) * 2));
        int x; int y = 3; int z = 1;
```

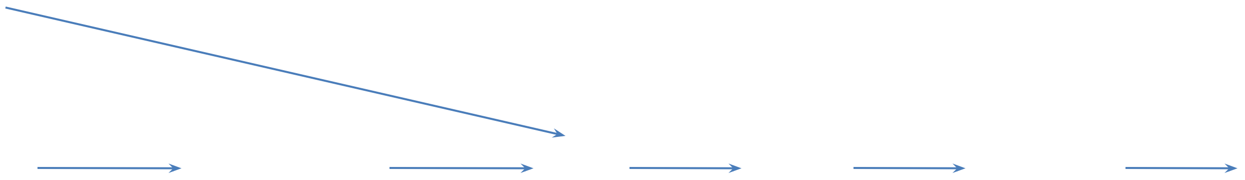
EXPRESSIONS

```
x = y = z;

System.out.println(" x = y = z: \t" + x);

System.out.println(" 3 - 2 + 1 = \t " + (3 − 2 + 1));

int i = 10; int j = 0; int result = 0;

result = i-- + i / 2 - ++i + j++ + ++j;

System.out.println("i: " +i+ " j " +j+ " result: "+result );

System.out.println("Hello "+1+2);

System.out.println(1+2+" Hello"); }}
```

EXPRESSIONS

PRIMITIVE TYPE CONVERSION AND CASTING

- In Java, type conversions are performed automatically when the type of the expression on the right-hand-side of an assignment operation can be safely promoted to the type of the variable on the left-hand-side of the assignment.
- Char
- byte short int long float double



EXPRESSIONS

- byte b = 10; // byte variable
- int i = b; // implicit widening byte to intType conversion or promotion also takes place while evaluating the expressions involving arithmetic operators.
- For example, int i = 10; //int variable
- double d = 20; //int literal assigned to a double variable
- d = i + d; //automatic conversion int to double

EXPRESSIONS

- For example, consider the following declarations: byte b = 10; short s = 30;
- The following statement is invalid because while evaluating the expression
- short z = b*s; //invalid
- int i = b*s; //valid

謝謝你
Danke
Gracias
Grazie
Merci
Thanks!
THANK YOU!

Sri Krishna Arts and Science College

Programming In Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

Classes and Objects

CLASSES :

- A class is a blueprint or prototype that defines the variables and methods common to all objects of a certain kind.
- In other words, a class can be thought of as a user-defined data type and an object as a variable of that data type that can contain data and methods, i.e., functions working on that data.

Classes and Objects

OBJECTS:

- An object is a software bundle that encapsulates variables and methods operating on those variables

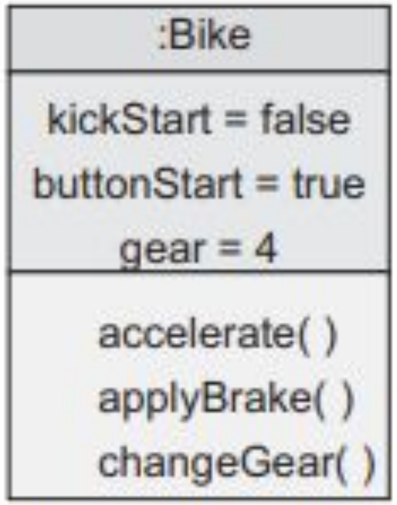


Fig. 4.2 Bike Object

Classes and Objects

CLASS DECLARATION IN JAVA

Syntax:

```
class Bike
{
    //Variables declaration //Methods declaration
}
```

Sample:

```
class GoodbyeWorld
 {
    public static void main (String args[])
    {
      System.out.println("Goodbye World!");
     }
 }
```

Classes and Objects

- The class declaration can specify more about the class, like you can:
- declare the superclass of a class
- list the interfaces implemented by the class
- declare whether the class is public, abstract, or fi nal
- class declaration in Java, we can summarize the class declaration syntax as
- [modifiers] class ClassName [extends SuperClassName] [implements InterfaceNames] { . . . }

Classes and Objects

- modifiers declare whether the class is public, protected, default, abstract or final
- ClassName sets the name of the class you are declaring
- SuperClassName is the name of the ClassName's superclass

Class Body:

- The class contains two different sections: variable declarations and method declarations.

Classes and Objects

There are three types of variables in Java:

- local variables, instance variables, and class variables.

- **Local Variables** are defined inside a method.

- **Instance Variable** is defined inside the class but outside the methods, and

- **Class Variables** are declared with the static modifier inside the class and outside the methods.

Classes and Objects

CREATING OBJECTS:

SalesTaxCalculator obj1 = new SalesTaxCalculator();

- This statement creates a new SalesTaxCalculator object.
- This single statement declares, instantiates, and initializes the object
- **Declaring an Object -**Object declarations are same as variable declarations.
- For example, SalesTaxCalculator obj1;
- Generally, the declaration is as follows:

type name

Classes and Objects

Instantiating an Object:

- The new operator instantiates a class by dynamically allocating (i.e., at runtime) memory for an object of the class.
- The new operator creates the object or instantiates an object and the constructor initializes it.
- SalesTaxCalculator obj1 = new SalesTaxCalculator()

Classes and Objects

Initializing an Object

- Initial values can be provided by instance variable initializers and constructors
- new SalesTaxCalculator()

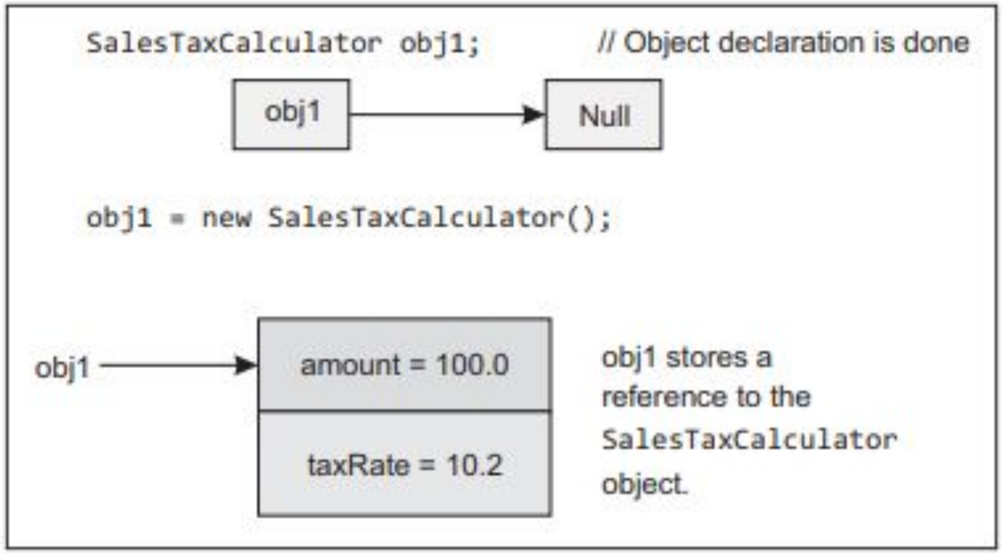

SalesTaxCalculator obj1;
// Object declaration is done
obj1
Null
obj1 = new SalesTaxCalculator();
obj1
amount = 100.0
taxRate = 10.2
obj1 stores a reference to the SalesTaxCalculator object.

Fig. 4.3 Steps in Object Creation

谢谢你
Grazie
Gracias
Danke
Merci
Thanks!
THANK YOU!

Sri Krishna Arts and Science College

Programming In Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

Agenda

- ✓ Classes
- ✓ Objects
- ✓ Methods
- ✓ Cleaning-up unused Objects.

Class in JAVA

What is a class in Java

A class is a group of objects which have common properties.

It is a template or blueprint from which objects are created.

It is a logical entity.

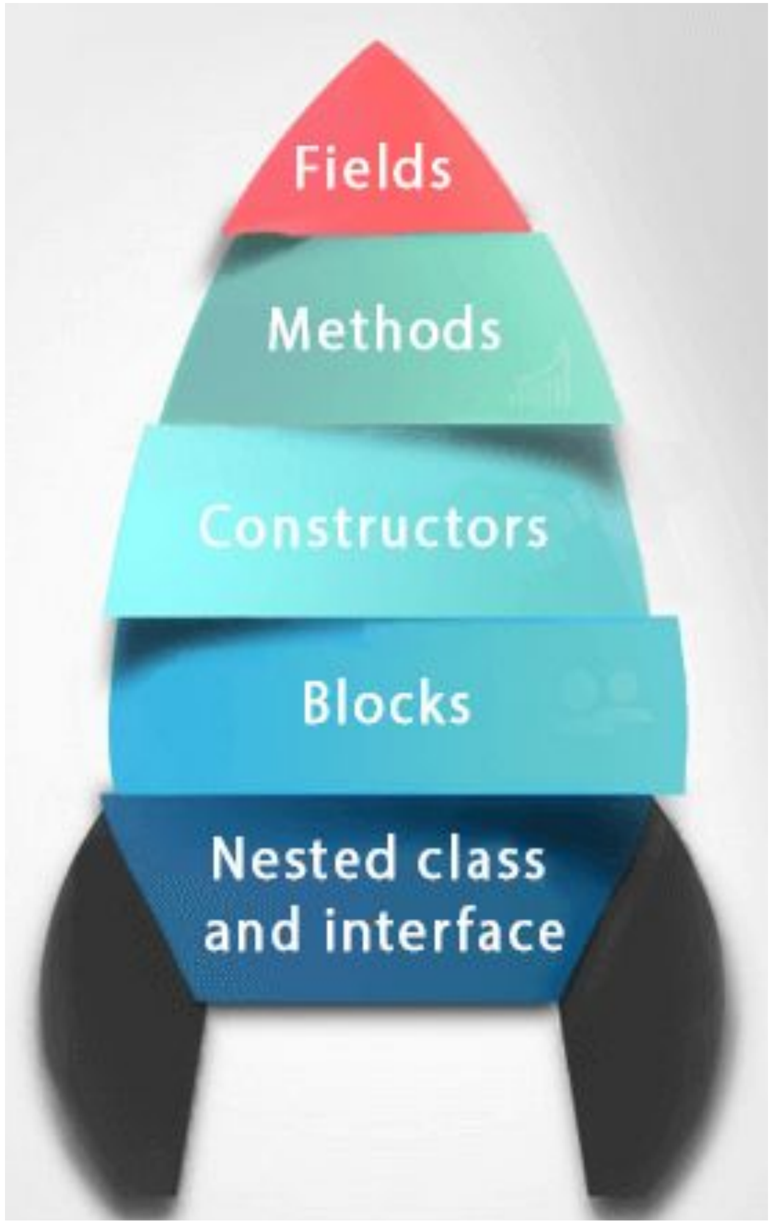
It can't be physical.

Class in JAVA

A class in Java can contain:

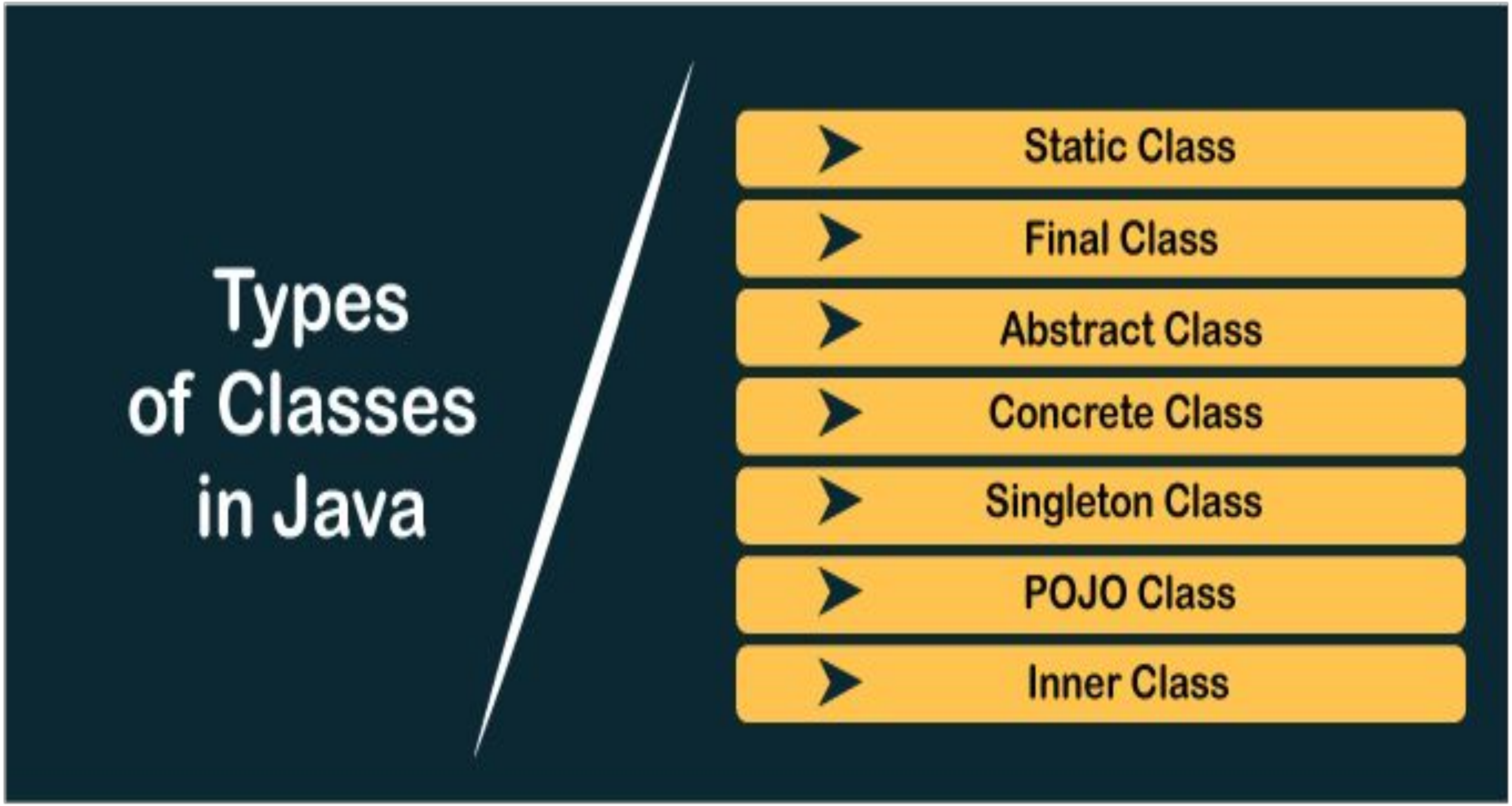
- ❑ Fields
- ❑ Methods
- ❑ Constructors
- ❑ Blocks
- ❑ Nested class and interface

Class in JAVA



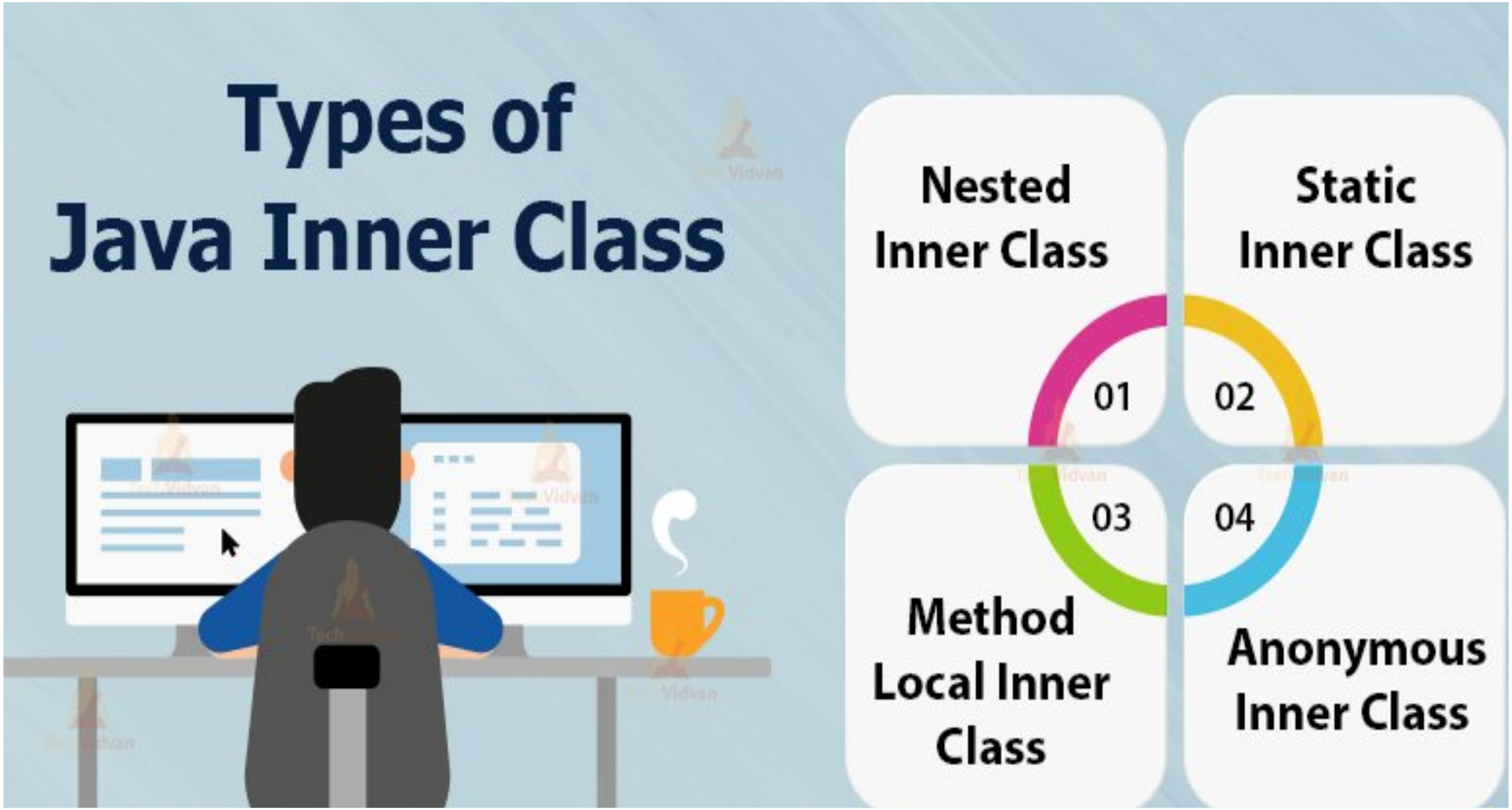
Syntax to declare a class:

```
class <class_name>
{
    field;
    method;
}
```



Types of Classes in Java

- Static Class
- Final Class
- Abstract Class
- Concrete Class
- Singleton Class
- POJO Class
- Inner Class


Types of
Java Inner Class
Nested Inner Class
01
Static Inner Class
02
03
Method Local Inner Class
04
Anonymous Inner Class

Objects in JAVA

What is an object in Java

- ✔ An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.
- ✔ It can be physical or logical (tangible and intangible).
- ✔ The example of an intangible object is the banking system.

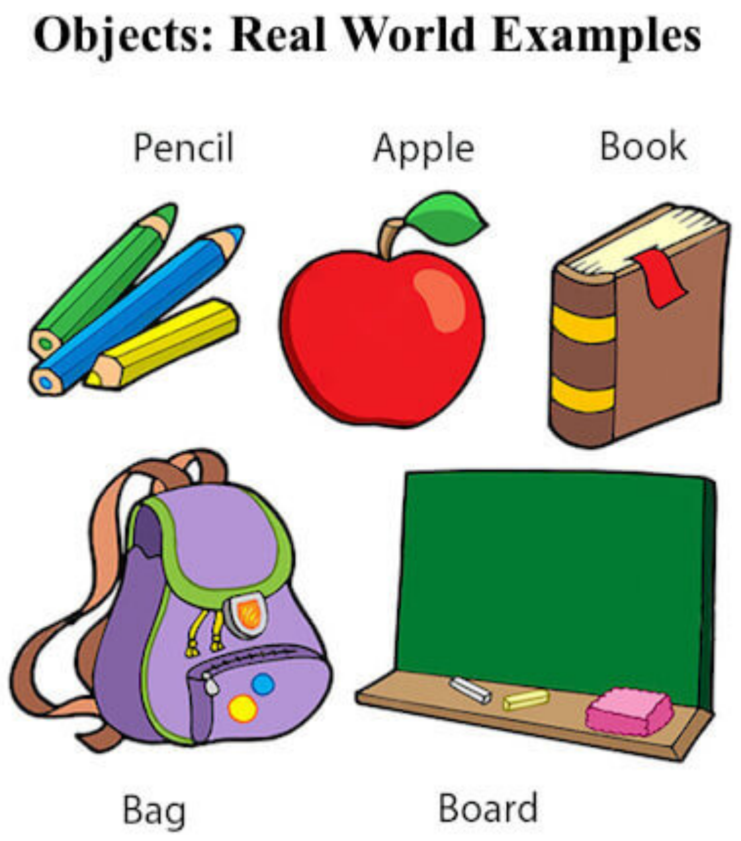
Objects in JAVA

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- ✔ An object is a real-world entity.
- ✔ An object is a runtime entity.
- ✔ The object is an entity which has state and behavior.
- ✔ The object is an instance of a class.

Objects in JAVA



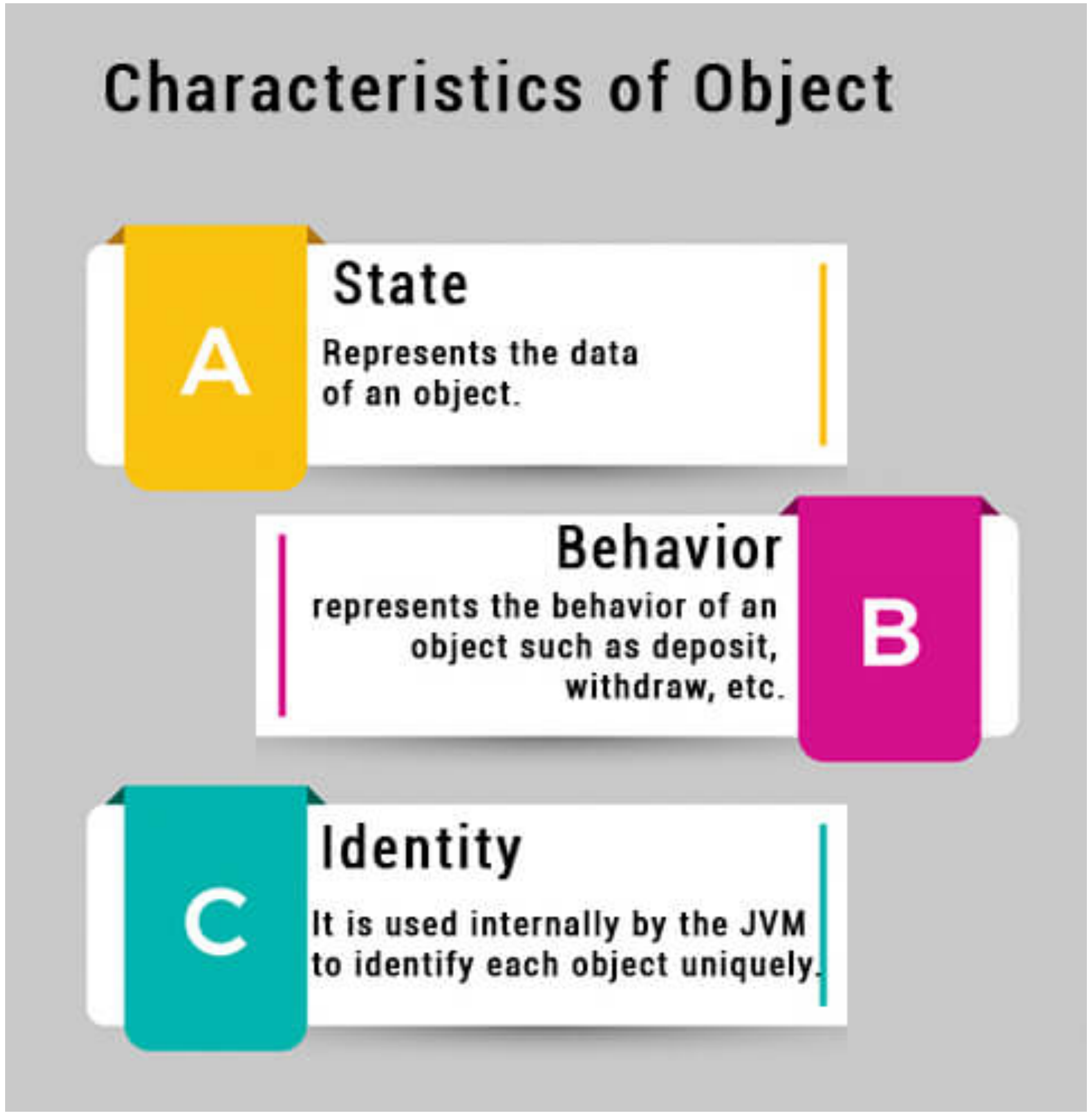
For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Objects Characteristics

- State: represents the data (value) of an object.
- Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Objects Characteristics


Characteristics of Object

A

State

Represents the data of an object.

Behavior

represents the behavior of an object such as deposit, withdraw, etc.

B

C

Identity

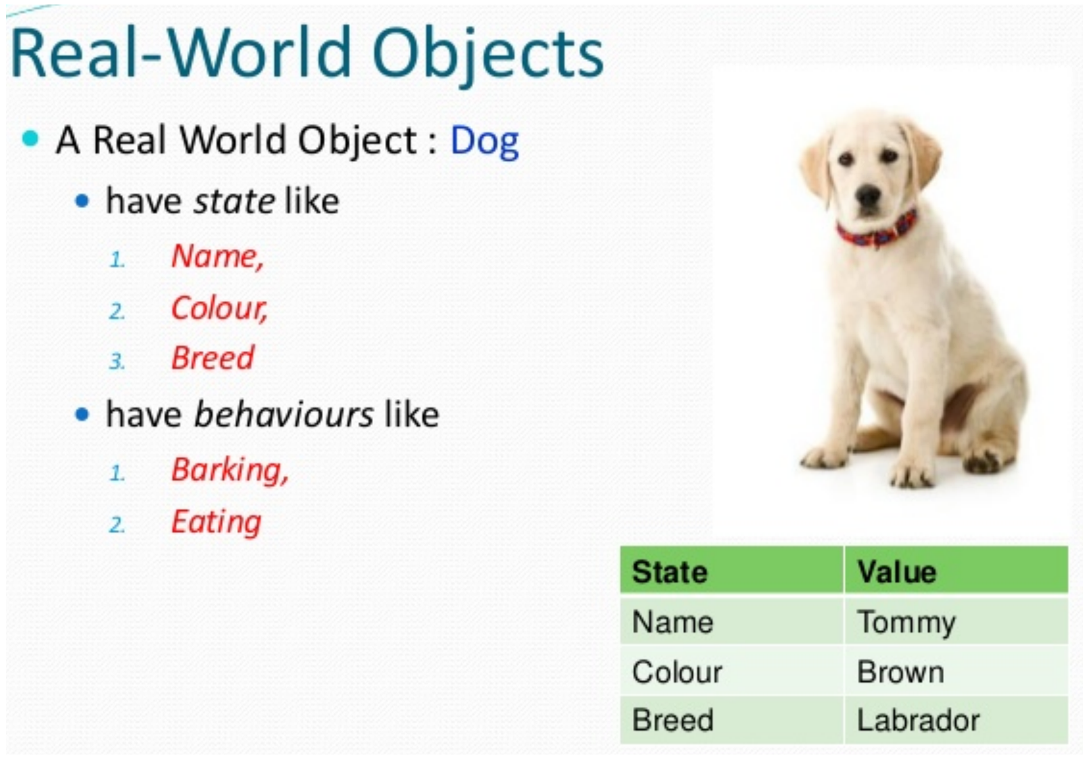
It is used internally by the JVM to identify each object uniquely.

Objects Characteristics Example

Identity
Name of dog

State/Attributes
Breed
Age
Color

Behaviors
Bark
Sleep
Eat

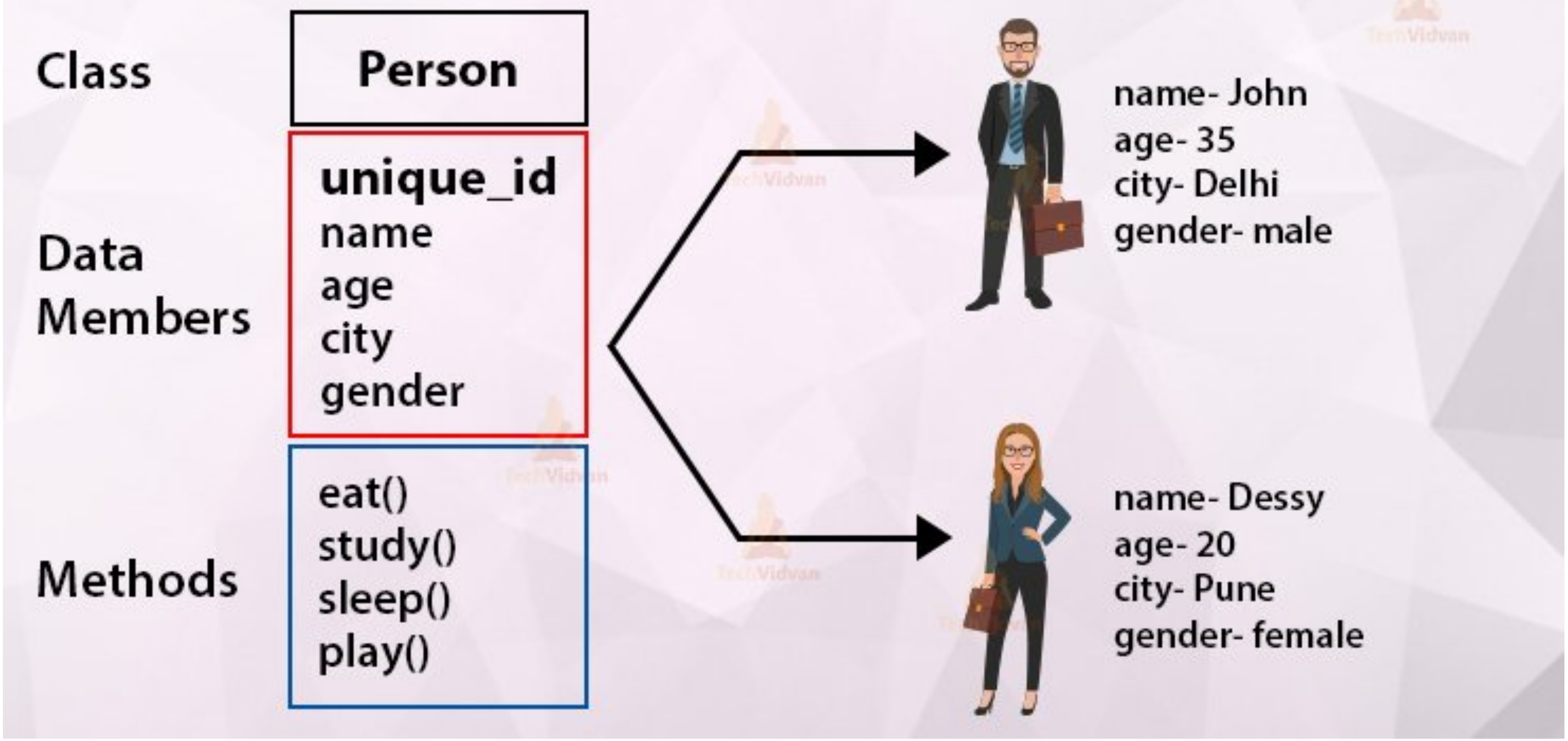


Real-World Objects

- A Real World Object : Dog
  - have *state* like
    1. *Name,*
    2. *Colour,*
    3. *Breed*
  - have *behaviours* like
    1. *Barking,*
    2. *Eating*

State	Value
Name	Tommy
Colour	Brown
Breed	Labrador

Java Class & Objects



Instance Variable in Java

- A variable which is created inside the class but outside the method is known as an instance variable.
- Instance variable doesn't get memory at compile time.
- It gets memory at runtime when an object or instance is created.
- That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- ✔ Code Reusability
- ✔ Code Optimization

New Keyword in Java

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Example Program for Class and Objects

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

Object Initialization in Java

3 Ways to initialize object

There are 3 ways to initialize object in Java.

- By reference variable
- By method
- By constructor

Object Initialization in Java

By reference variable

Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

Object Initialization in Java

By reference variable : Example Program

```
class Student
{
 int id;
 String name;
}
class TestStudent2
{
 public static void main(String args[])
{
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);
}
}
```

Object Initialization in Java

By method

Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method.

Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

Object Initialization in Java

By method : Example

```
class Student
{
 int rollno;
 String name;
 void insertRecord(int r, String n)
{
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);
}
}
```

Object Initialization in Java

By method : Example

```
class TestStudent4
{
public static void main(String args[])
{
 Student s1=new Student();
 Student s2=new Student();
 s1.insertRecord(111,"Karan");
 s2.insertRecord(222,"Aryan");
 s1.displayInformation();
 s2.displayInformation();
}
}
```

Object Initialization in Java

By constructor

Object and Class Example: Initialization through a constructor

```
class Employee
{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s)
{
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);
}
}
```

Object Initialization in Java

By constructor

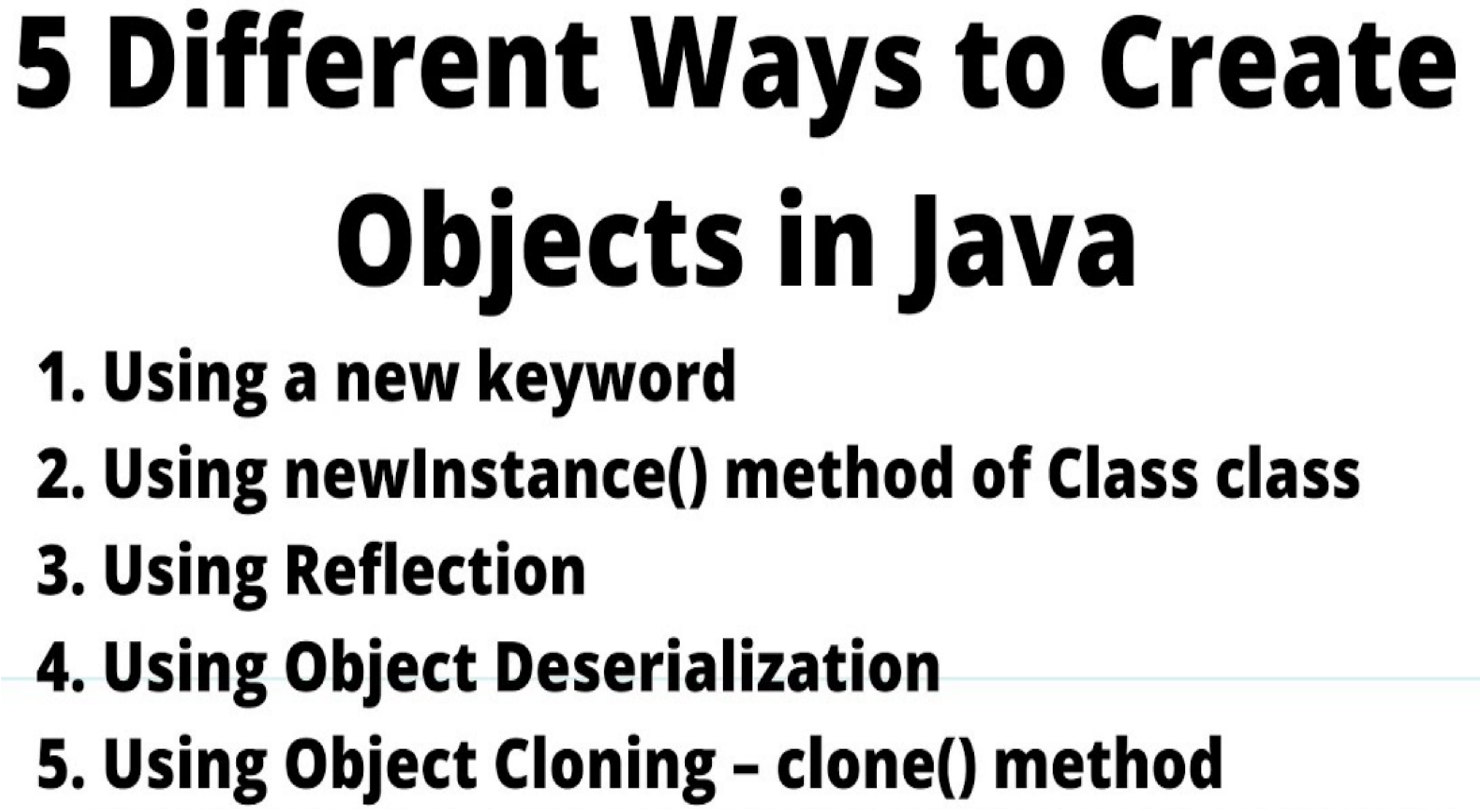
Object and Class Example: Initialization through a constructor

```
public class TestEmployee
{
public static void main(String[] args)
 {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```

Object & Class Example Program

```
class Rectangle
{
 int length;
 int width;
 void insert(int l, int w)
{
  length=l;
  width=w;
 }
 void calculateArea()
{
System.out.println(length*width);
}
}
```

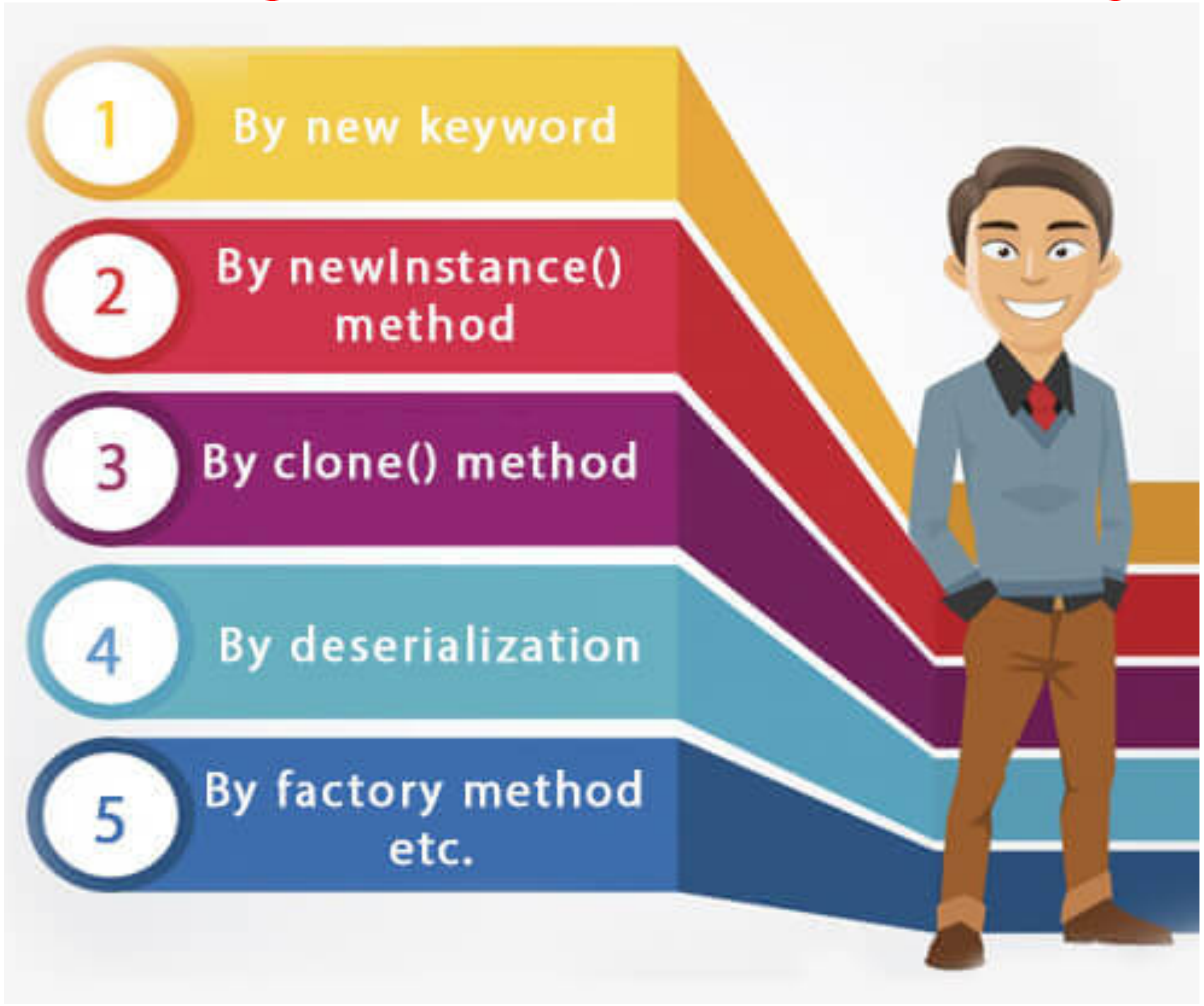
```
class TestRectangle1
{
 public static void main(String args[])
{
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```



5 Different Ways to Create Objects in Java

1. Using a new keyword
2. Using newInstance() method of Class class
3. Using Reflection
4. Using Object Deserialization
5. Using Object Cloning – clone() method

Different Ways to Create an Object in Java



Cleaning up Unused Objects

Cleaning Up Unused Objects

- ❖ Some object-oriented languages require that you keep track of all the objects you create, and that you explicitly destroy them when they are no longer needed.
- ❖ Managing memory explicitly is tedious and error-prone.
- ❖ The Java platform allows you to create as many objects as you want (limited, of course, by what your system can handle), and you don't have to worry about destroying them.
- ❖ The Java runtime environment deletes objects when it determines that they are no longer being used.
- ❖ This process is called *garbage collection*.

Cleaning up Unused Objects

Cleaning Up Unused Objects

- ❖ An object is eligible for garbage collection when there are no more references to that object.
- ❖ References that are held in a variable are usually dropped when the variable goes out of scope.
- ❖ Or, you can explicitly drop an object reference by setting the variable to the special value null.
- ❖ Remember that a program can have multiple references to the same object; all references to an object must be dropped before the object is eligible for garbage collection.

Cleaning up Unused Objects

The Garbage Collector

- ❖ The Java runtime environment has a garbage collector that periodically frees the memory used by objects that are no longer referenced.
- ❖ The garbage collector does its job automatically, although in some situations, you may want to explicitly request garbage collection by invoking the gc method in the System class.
- ❖ For example, you might want to do this after a section of code that creates a large amount of garbage, or before a section of code that needs a lot of memory.
- ❖ In most situations, however, it is enough to simply let the runtime environment run the garbage collector on its own, when it determines that the time is right.

Cleaning up Unused Objects

Finalization

- Before an object gets garbage-collected, the garbage collector gives the object an opportunity to clean up after itself through a call to the object's finalize method.
- This process is known as *finalization*.
- Most programmers don't have to worry about implementing the finalize method.
- In rare cases, however, a programmer might have to implement a finalize method to release resources, such as native peers, that aren't under the control of the garbage collector.

Cleaning up Unused Objects

Finalization

- ❖ The finalize method is a member of the Object class, which is the top of the Java platform's class hierarchy.
- ❖ It is the superclass of all classes.
- ❖ A class can override the finalize method to perform any finalization necessary for objects of that type.
- ❖ If you override finalize, your implementation of the method should call super. finalize as the last thing that it does.
- ❖ Overriding and Hiding Methods talks more about how to override methods.

Thank You

Sri Krishna Arts and Science College

Programming In Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

Agenda

- Class Variables
- Class Method
- Static Keyword

Class Variables

A variable provides us with named storage that our programs can manipulate. Java provides three types of variables.

1. **Class Variables**
2. **Instance Variables**
3. **Local Variables**

Class Variables

Class variables

- ✔ Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- ✔ There would only be one copy of each class variable per class, regardless of how many objects are created from it.

Class / Static Variables

- ✔ Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- ✔ There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- ✔ Static variables are rarely used other than being declared as constants.

Class / Static Variables

- ✔ Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- ✔ Static variables are stored in the static memory.
- ✔ It is rare to use static variables other than declared final and used as either public or private constants.
- ✔ Static variables are created when the program starts and destroyed when the program stops.

Class / Static Variables

- ✔ Visibility is similar to instance variables.
- ✔ However, most static variables are declared public since they must be available for users of the class.
- ✔ Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null.

Class / Static Variables

- ✔ Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- ✔ Static variables can be accessed by calling with the class name ClassName.VariableName.
- ✔ When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Class / Static Variables : Example

```
import java.io.*;
public class Employee
{
   // salary  variable is a private static variable
   private static double salary;
   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";
   public static void main(String args[])
{
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + salary);
   }
```

Local Variables

- ✔ Local variables are declared in methods, constructors, or blocks.
- ✔ Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- ✔ Access modifiers cannot be used for local variables.
- ✔ Local variables are visible only within the declared method, constructor, or block.
- ✔ Local variables are implemented at stack level internally.
- ✔ There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Local Variables : Example

```
public class Test
{
   public void pupAge()
{
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }
   public static void main(String args[])
 {
      Test test = new Test();
      test.pupAge();
   }
}
```

Here, age is a local variable.
This is defined inside pupAge() method and its scope is limited to only this method.

Instance Variables

- ✔ Instance variables are declared in a class, but outside a method, constructor or any block.
- ✔ When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- ✔ Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- ✔ Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- ✔ Instance variables can be declared in class level before or after use.

Instance Variables

- ✔ Access modifiers can be given for instance variables.
- ✔ The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- ✔ Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

Local Variables : Example

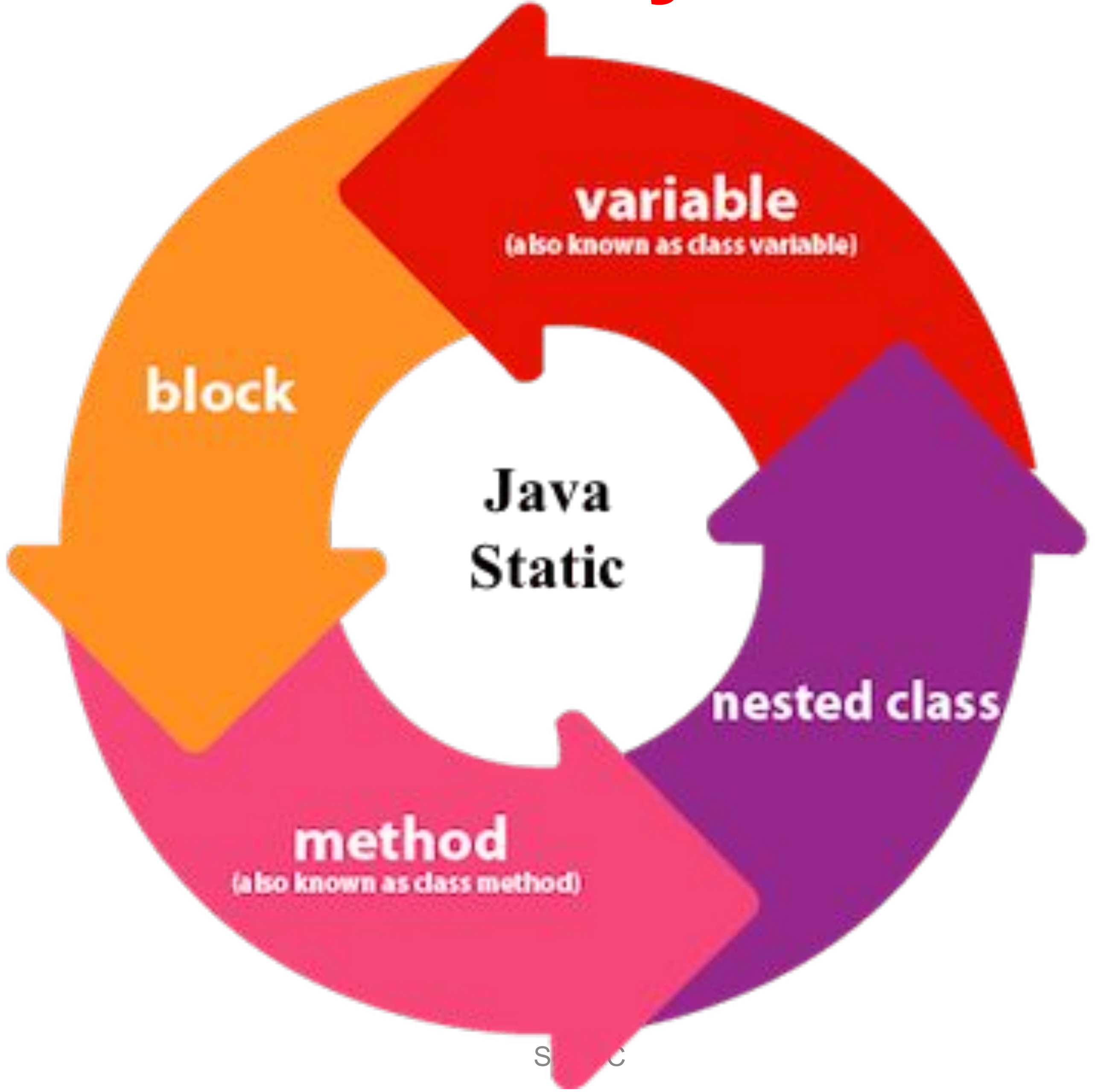
```
import java.io.*;
public class Employee
{
  // this instance variable is visible for any child class.
  public String name;
  // salary  variable is visible in Employee class only.
  private double salary;
  // The name variable is assigned in the constructor.
  public Employee (String empName)
{
    name = empName;
  }
  // The salary variable is assigned a value.
  public void setSalary(double empSal)
{
    salary = empSal;
  }
  // This method prints the employee details.
  public void printEmp() {
    System.out.println("name  : " + name );
    System.out.println("salary :" + salary);
  }
  public static void main(String args[])
{
    Employee empOne = new Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
  }}
```

Static Keyword

- ✔ The **static keyword** in Java is used for memory management mainly.
- ✔ We can apply static keyword with variables, methods, blocks and nested classes.
- ✔ The static keyword belongs to the class than an instance of the class.
- ✔ **The static can be:**

  1. **Variable (also known as a class variable)**
  2. **Method (also known as a class method)**
  3. **Block**
  4. **Nested class**

Static Keyword



1. Static Variable

Java static variable

- ✔ If you declare any variable as static, it is known as a static variable.
- ✔ The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- ✔ The static variable gets memory only once in the class area at the time of class loading.
- ✔ **Advantages of static variable**
  - ✔ It makes your program memory efficient (i.e., it saves memory).

Without Static Variable

Understanding the problem without static variable

```
class Student
{
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Example With Static Variable

```
//Java Program to demonstrate the use of static variable
class Student
{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   Student(int r, String n)
{
   rollno = r;
   name = n;
   }
   void display ()
{
System.out.println(rollno+" "+name+" "+college);
}
}
```

Example With Static Variable

```
//Test class to show the values of objects
public class TestStaticVariable1
{
 public static void main(String args[])
{
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
```

2. Static Method

- ✔ When a method is declared with the static keyword, it is known as the static method.
- ✔ The most common example of a static method is the main( ) method.
- ✔ As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.
- ✔ **Methods declared as static have several restrictions:**
    - ✔ They can only directly call other static methods.
    - ✔ They can only directly access static data.
    - ✔ They cannot refer to this or super in any way.

2. Static Method

Java static method

- ✔ If you apply static keyword with any method, it is known as static method.
- ✔ A static method belongs to the class rather than the object of a class.
- ✔ A static method can be invoked without the need for creating an instance of a class.
- ✔ A static method can access static data member and can change the value of it.

Example of Static Method

```
//Java Program to demonstrate the use of a static method.
class Student
{
     int rollno;
     String name;
     static String college = "ITS";
     //static method to change the value of static variable
     static void change()
{
     college = "BBDIT";
     }
Student(int r, String n)
{
     rollno = r;
     name = n;
     }
void display()
{
System.out.println(rollno+" "+name+" "+college);
}  }
```

Example of Static Method

```
//Test class to create and display the values of object
public class TestStaticMethod
{
   public static void main(String args[])
{
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
   }
}
```

3. Static Block

Java static block

- ✔ Is used to initialize the static data member.
- ✔ It is executed before the main method at the time of class loading.

Example of static block

```
class A2
{
  static
{
System.out.println("static block is invoked");
}
  public static void main(String args[])
{
   System.out.println("Hello main");
  }
}
```

4. Nested Class / Static Class

- ✔ A class can be made **static** only if it is a nested class.
- ✔ We cannot declare a top-level class with a static modifier but can declare nested classes as static.
- ✔ Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class.
- ✔ In this case, a static class cannot access non-static members of the Outer class.

Example of Nested Class / Static Class

```
import java.io.*;
public class GFG
{
      private static String str = "GeeksforGeeks";
    // Static class
    static class MyNestedClass
{
        // non-static method
        public void disp()
{
          System.out.println(str);
        }
    }
    public static void main(String args[])
    {
        GFG.MyNestedClass obj = new GFG.MyNestedClass();
        obj.disp();
    }}
```

Thank You

Sri Krishna Arts and Science College

Programming In Java

S.Vetrivel

Assistant Professor

ICT & Cognitive Systems

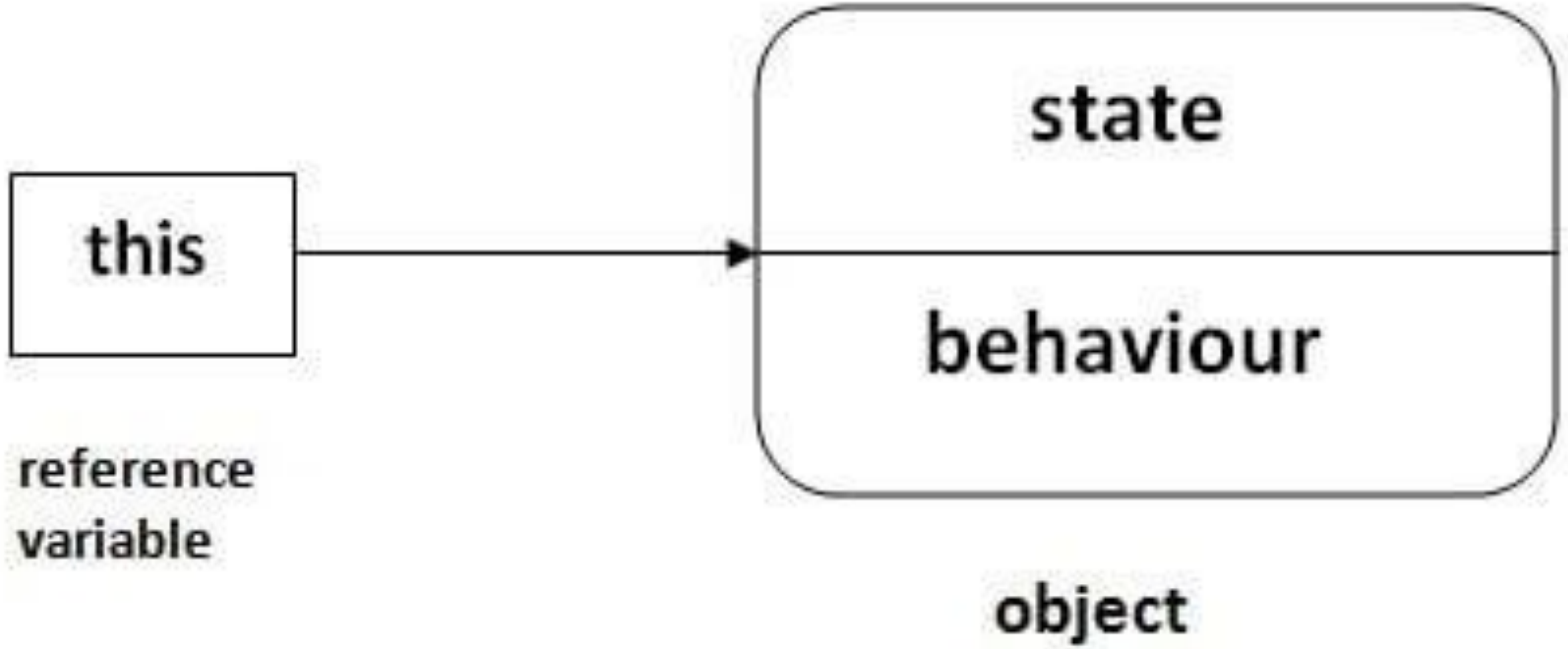
Agenda

- **this Keyword**

this Keyword

- The this keyword refers to the current object in a method or constructor.
- The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

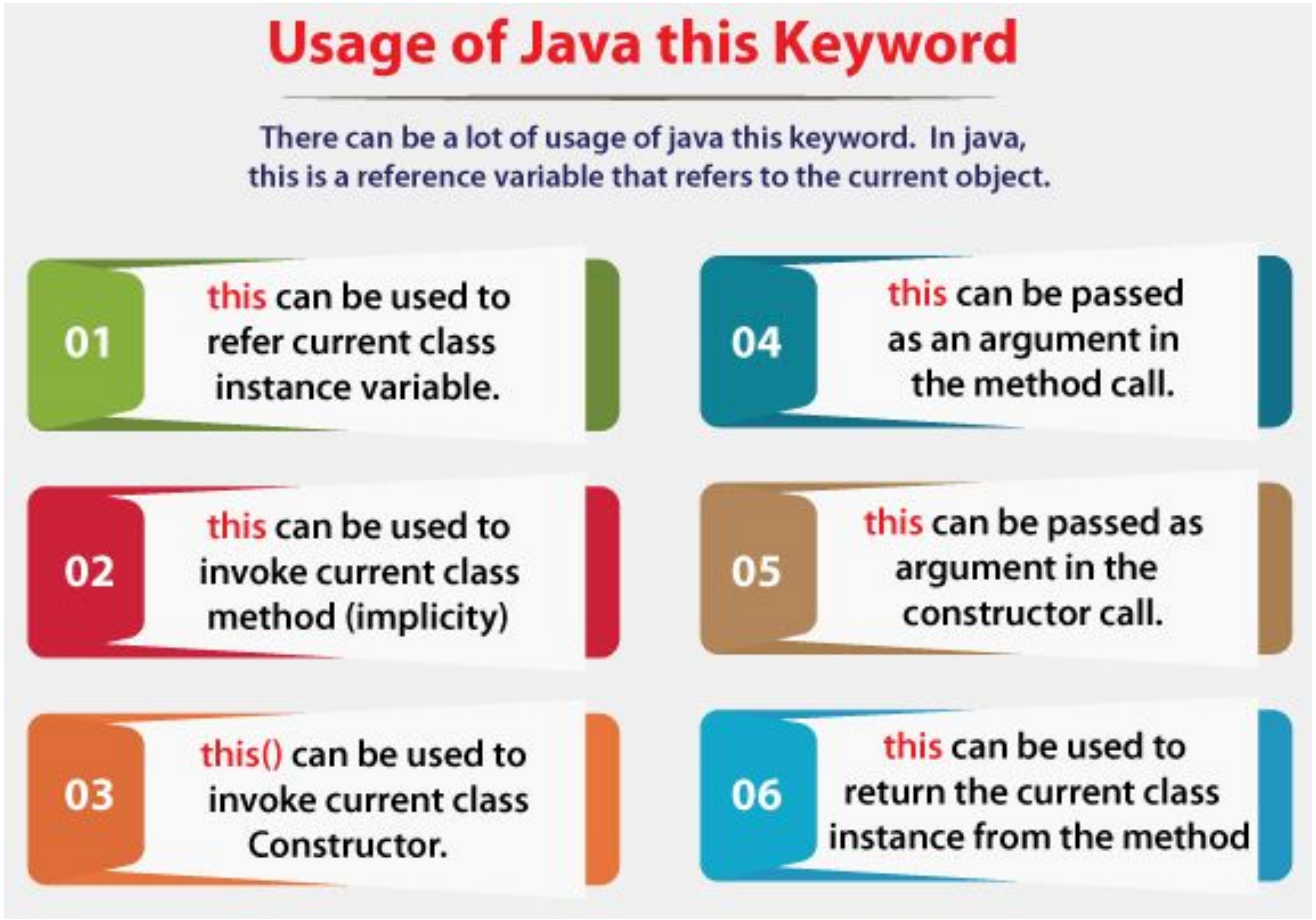
this Keyword



this Keyword

Usage of Java this keyword

1. Here is given the 6 usage of java this keyword.
2. this can be used to refer current class instance variable.
3. this can be used to invoke current class method (implicitly)
4. this() can be used to invoke current class constructor.
5. this can be passed as an argument in the method call.
6. this can be passed as argument in the constructor call.
7. this can be used to return the current class instance from the method.



Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01 this can be used to refer current class instance variable.

02 this can be used to invoke current class method (implicity)

03 this() can be used to invoke current class Constructor.

04 this can be passed as an argument in the method call.

05 this can be passed as argument in the constructor call.

06 this can be used to return the current class instance from the method

this Keyword

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Without this Keyword

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
rollno=rollno;
name=name;
fee=fee;
}
void display()
{
System.out.println(rollno+" "+name+" "+fee);
}
}
```

```
class TestThis1
{
public static void main(String args[])
{
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

this Keyword

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display()
{
System.out.println(rollno+" "+name+"
"+fee);
}
}
```

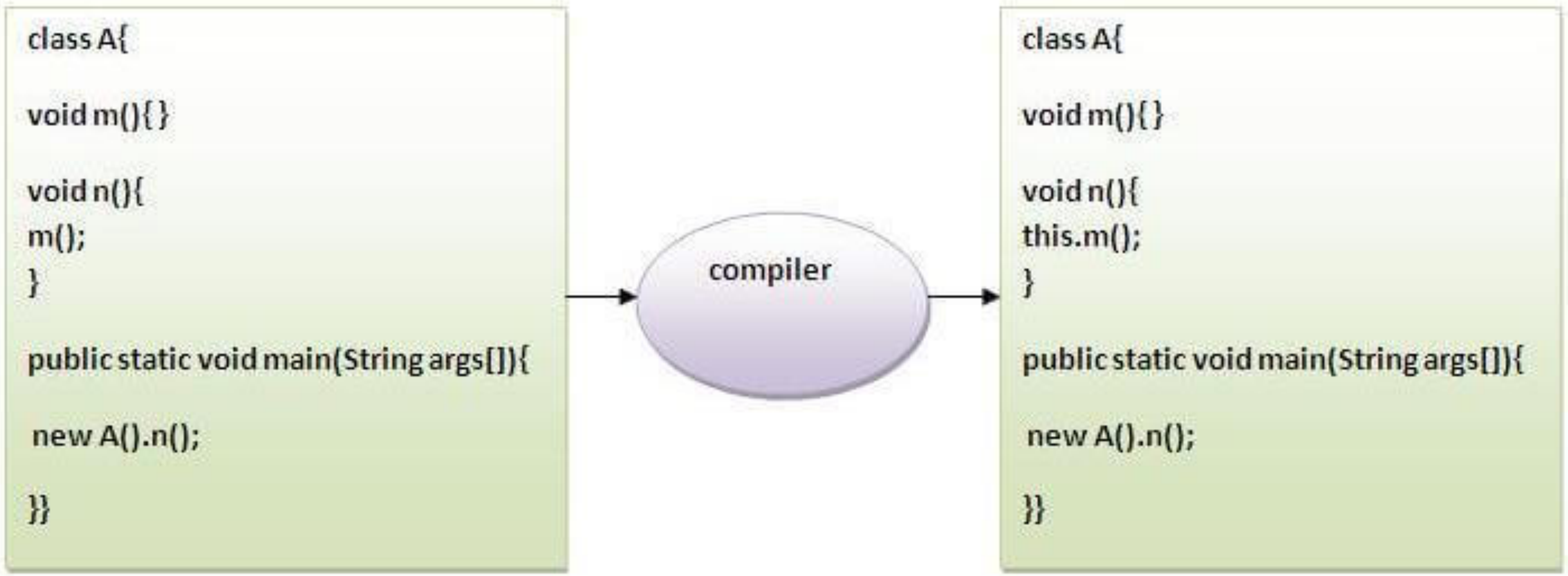
```
class TestThis2
{
public static void main(String args[])
{
Student s1=new
Student(111,"ankit",5000f);
Student s2=new
Student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

this Keyword

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

this Keyword



```
class A{

void m(){}

void n(){
m();
}

public static void main(String args[]){

 new A().n();

}}
```

```
class A{

void m(){}

void n(){
this.m();
}

public static void main(String args[]){

 new A().n();

}}
```

Example

```
class A
{
void m()
{
System.out.println("hello m");
}
void n()
{
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
```

```
class TestThis
{
public static void main(String args[])
{
A a=new A();
a.n();
}
}
```

this Keyword

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

this Keyword

```
class A
{
A()
{
System.out.println("hello a");
}
A(int x)
{
this();
System.out.println(x);
}
}
```

```
class TestThis
{
public static void main(String args[])
{
A a=new A(10);
}
}
```

this Keyword

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method.

It is mainly used in the event handling.

Let's see the example:

this Keyword

```
class S2
{
  void m(S2 obj)
{
  System.out.println("method is invoked");
  }
  void p()
{
  m(this);
  }
  public static void main(String args[])
{
  S2 s1 = new S2();
  s1.p();
  }
}
```

this Keyword

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also.

It is useful if we have to use one object in multiple classes.

Let's see the example:

this Keyword

```
class B
{
  A4 obj;
  B(A4 obj)
{
    this.obj=obj;
  }
  void display()
{
System.out.println(obj.data);
  }
}
```

```
class A4
{
  int data=10;
  A4()
{
   B b=new B(this);
   b.display();
  }
  public static void main(String args[])
{
   A4 a=new A4();
  }
}
```

this Keyword

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method.

In such case, return type of the method must be the class type (non-primitive).

Let's see the example:

this Keyword

```
class A
{
A getA()
{
return this;
}
void msg()
{
System.out.println("Hello java");
}
}
```

```
class Test1
{
public static void main(String args[])
{
new A().getA().msg();
}
}
```

Thank You