JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. **Classes and Instances**
2. The Methods
3. The Constructors
4. Static Elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and Interfaces
9. String processing
10. Wrapper classes for primitive types
11. Exceptions and Assertions
12. Nested classes
13. Enums
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java classes
19. Object Oriented Design
20. Functional Programming

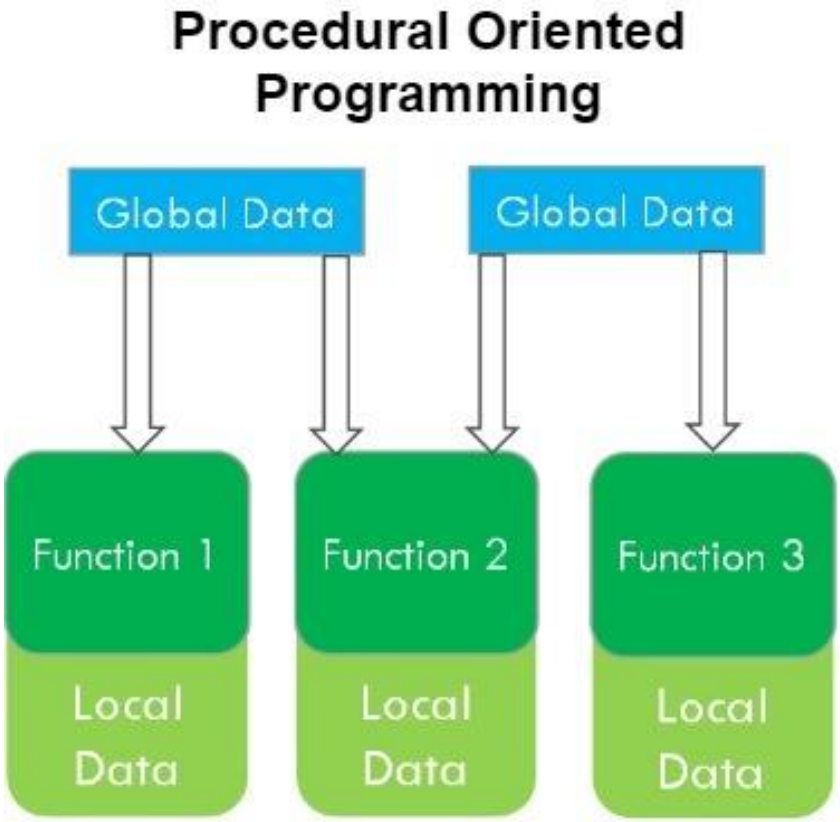
Module contents

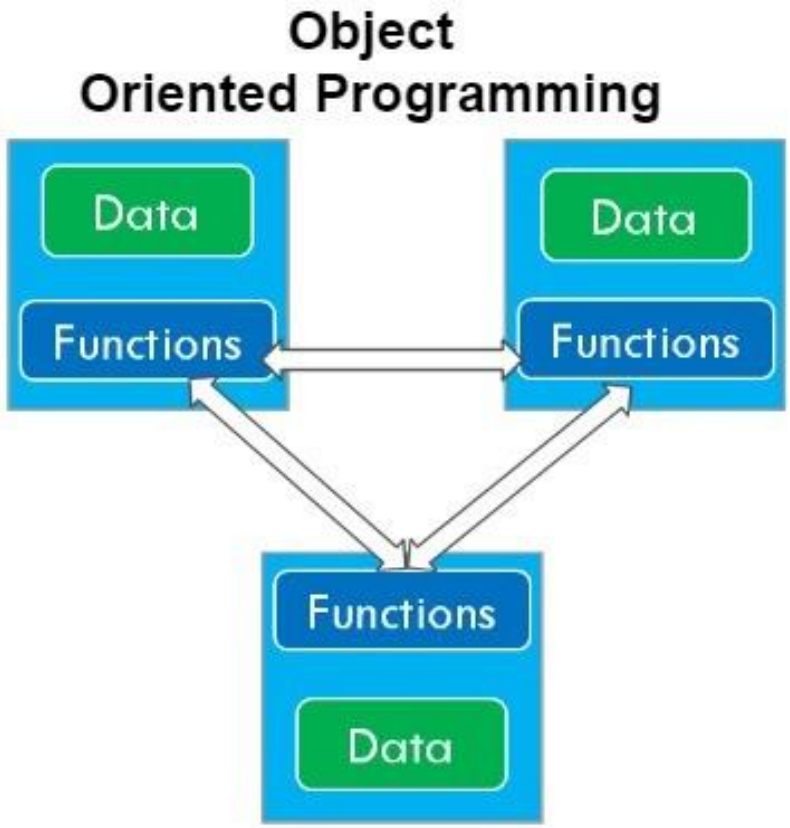
- Classes and Instances
  - Overview of class declarations. Class Fields and Methods.
  - Access modifiers
  - Encapsulation
  - Creating Objects
  - null literal
  - this Keyword

Module contents

- Classes and Instances
  - Overview of class declarations. Class Fields and Methods.
  - Access modifiers
  - Encapsulation
  - Creating Objects
  - null literal
  - this Keyword

Procedural vs Object-Oriented Programing





The task of calculating the area of a room free from furniture.

Procedural vs Object-Oriented Programing

The task of calculating the area of a room free from furniture.

What Is a Class?

- A class is a blueprint or prototype from which objects are created.
- A class models the state and behavior of a real-world object and class can cleanly model state and behavior.

What Is an Object?

- An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects.

- An object is a software bundle of variables and related methods.
- Objects are related to real life scenario
- Class is the general thing and object is the specialization of general thing
- Objects are instance of classes.

- An object is characterized by concepts like:

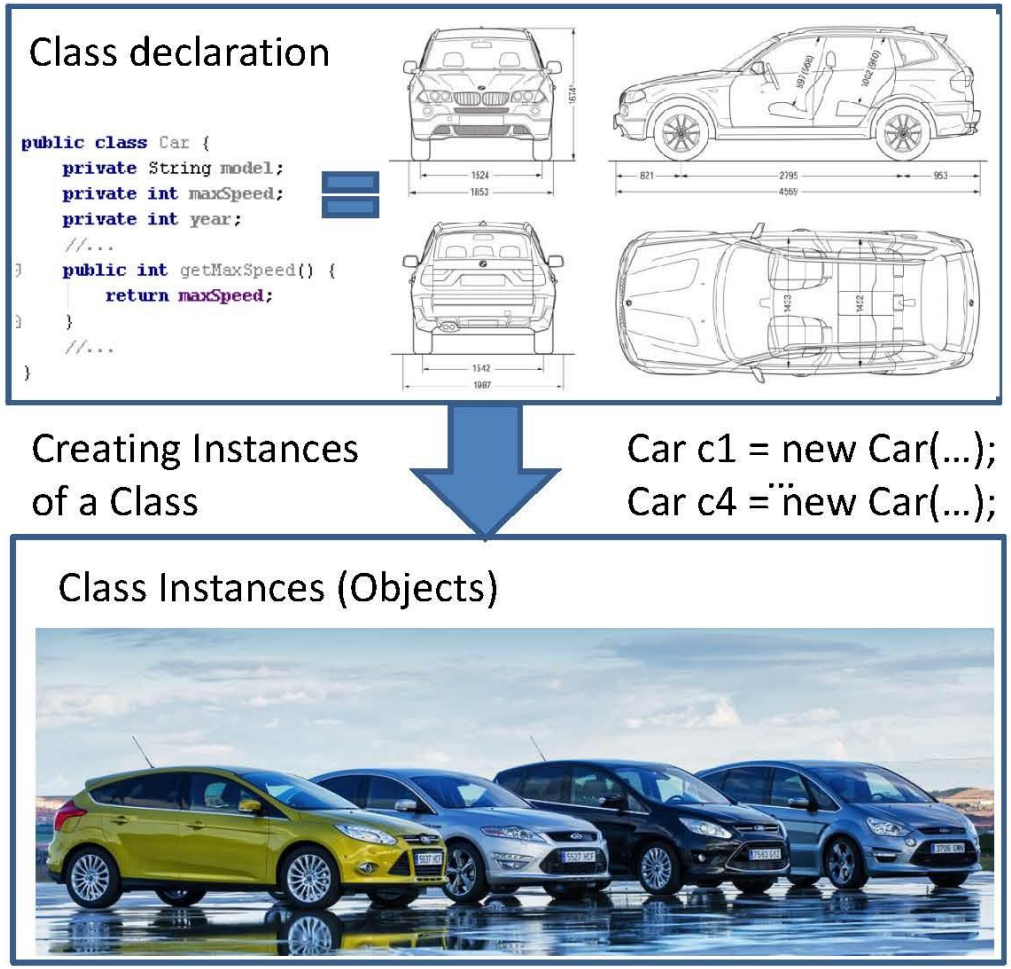
-Attribute

-Behavior

-Identity

Classes and Instances

- "Class" means a category of things
- – A class name can be used as the type of a field or local variable
- "Object" means a particular item that belongs to a class
- – Also called an "instance"



Class Fields and Methods

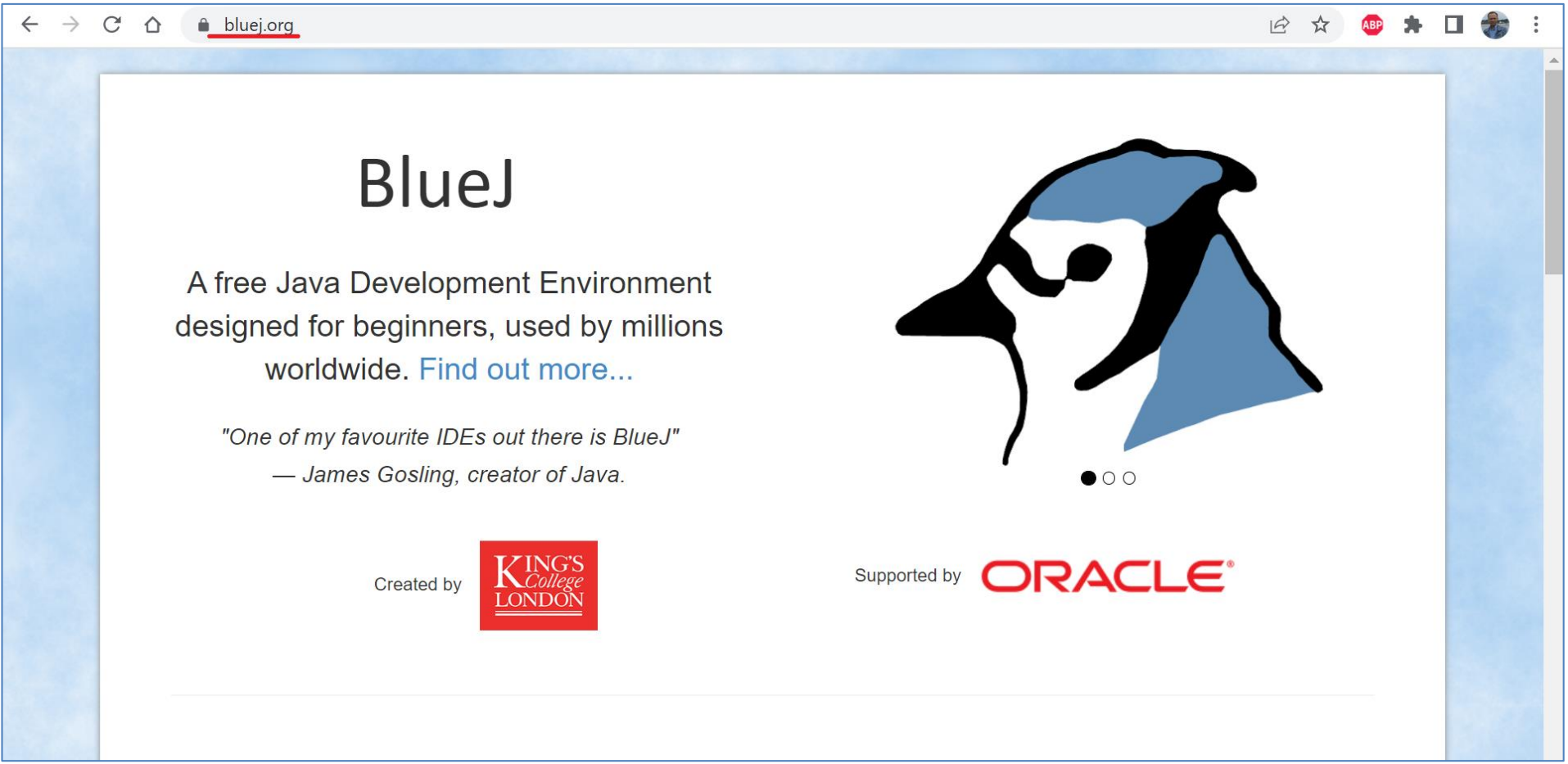
[ClassModifiers] **class** **ClassIdentifier** [<TypeParameters>]
[extends SuperClass] [implements Interfaces]

- Package
- Fields
- Constructor
- Methods

```
1.  package com.brainacad.oop1;
2.  public class Car {
3.      private String model;
4.      private int maxSpeed;
5.      private int year;
6.      private int speed;
7.      //...
        public Car(String model,int year){
8.          this.model = model;
9.          this.year = year;
        }
10.     //...
11.     public int getMaxSpeed() {
12.         return maxSpeed;
13.     }
        public int getSpeed() {
14.         return speed;
15.     }
        //...
16. }
```

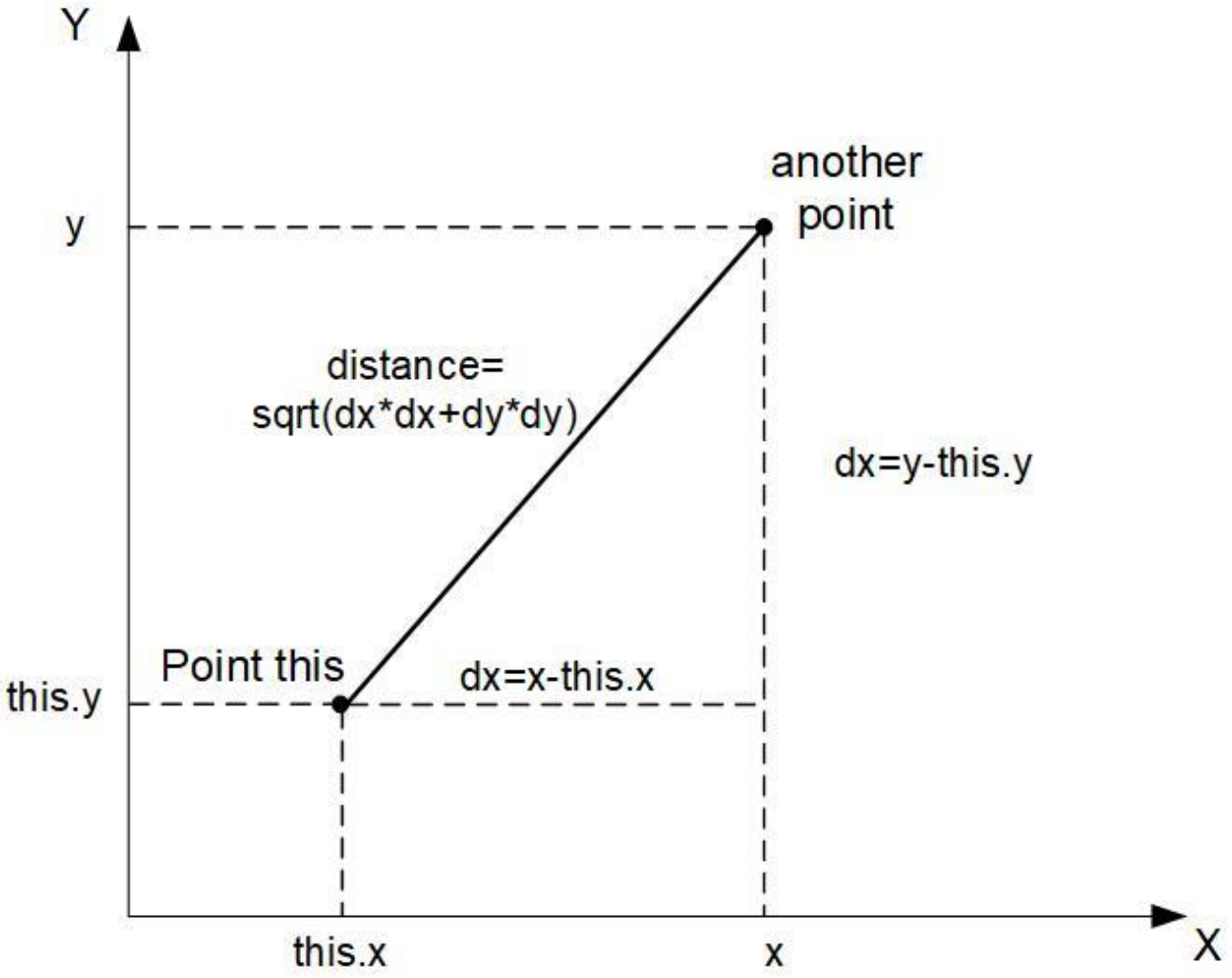
Class declaration

BlueJ



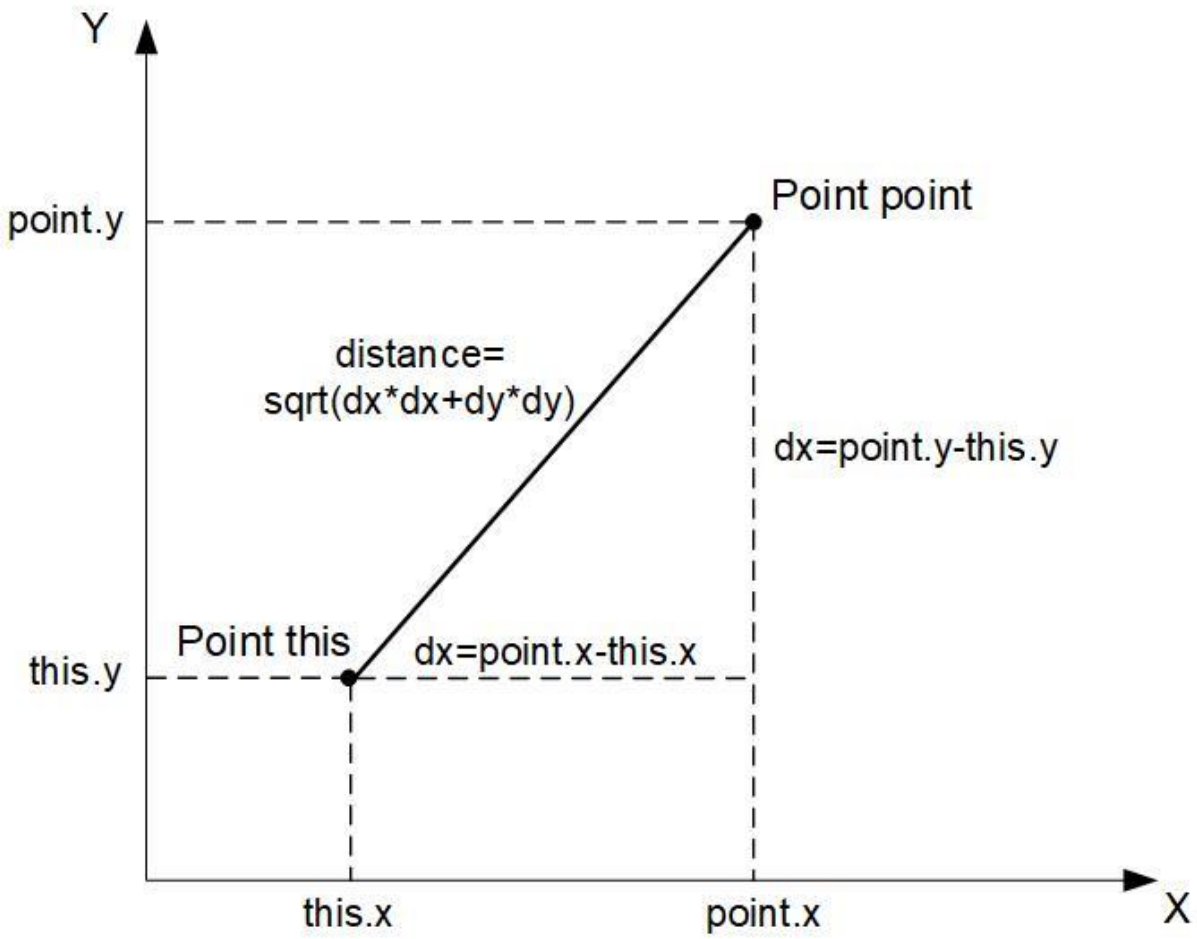
BlueJ was developed to support the learning and teaching of object-oriented programming. The main screen graphically shows the class structure of an application under development (in a UML-like diagram), and objects can be interactively created and tested.

BlueJ - Point on the plane - 1



See BJPoint1

BlueJ - Point on the plane - 2



Module contents

- Classes and Instances
  - Overview of class declarations. Class Fields and Methods.
  - Access modifiers
  - Encapsulation
  - Creating Objects
  - null literal
  - this Keyword

Access modifiers

At top-level (at class, interface, enum level):

1. public
2. default (not actually an access specifier)

See classmembersvisibility

At members of class level:

1. public
2. protected
3. default (not actually an access specifier)
4. private

default ≡ package-private

nested classes may be private and default too

Access modifiers 1/2

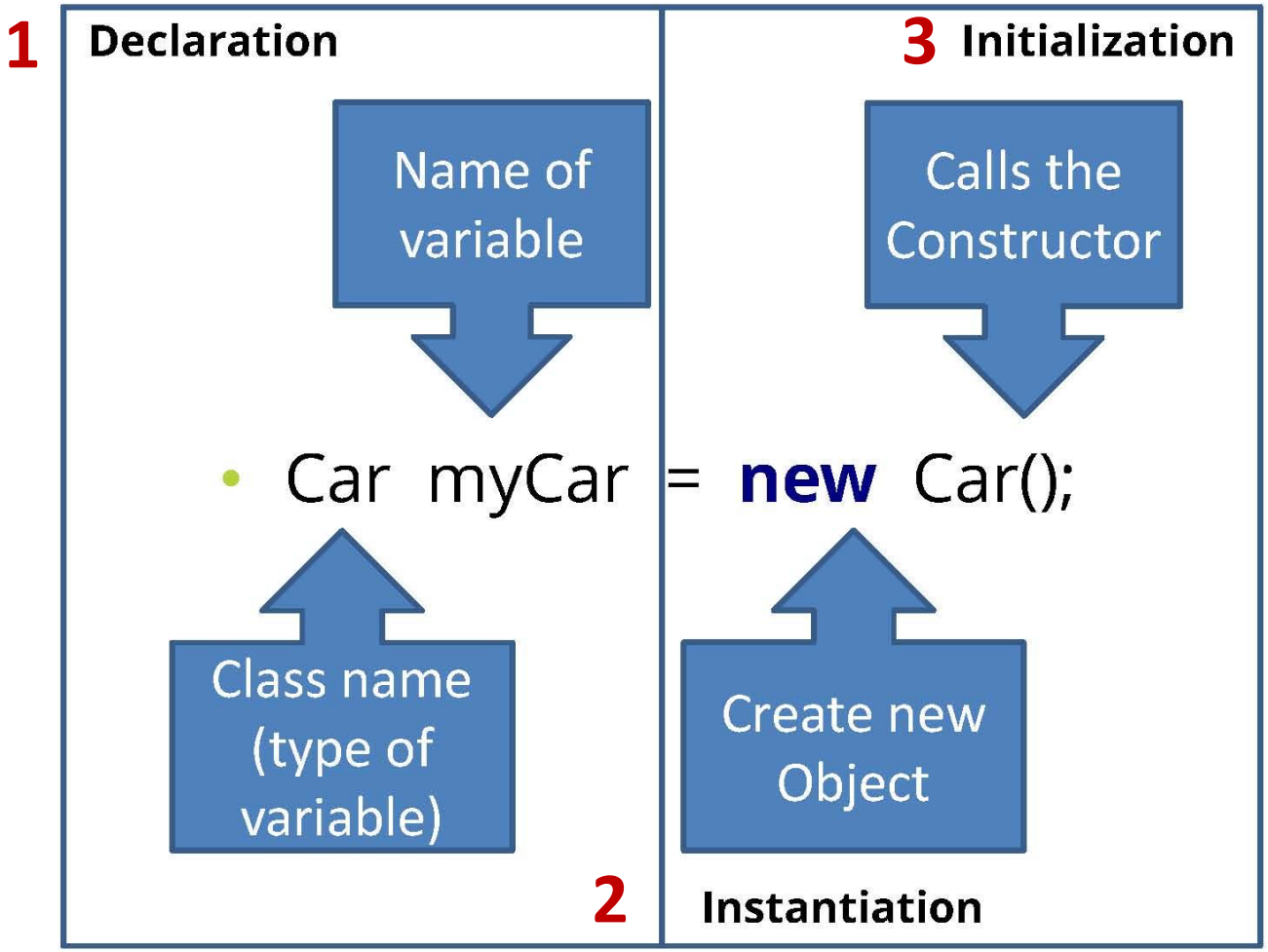
Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, through inheritance	No	No
From any non-subclass outside the package	Yes	No	No	No

Local variables and method parameters can’t be defined using access modifiers

Module contents

- Classes and Instances
  - Overview of class declarations. Class Fields and Methods.
  - Access modifiers
  - Encapsulation
  - Creating Objects
  - null literal
  - this Keyword

Creating Objects 1/3

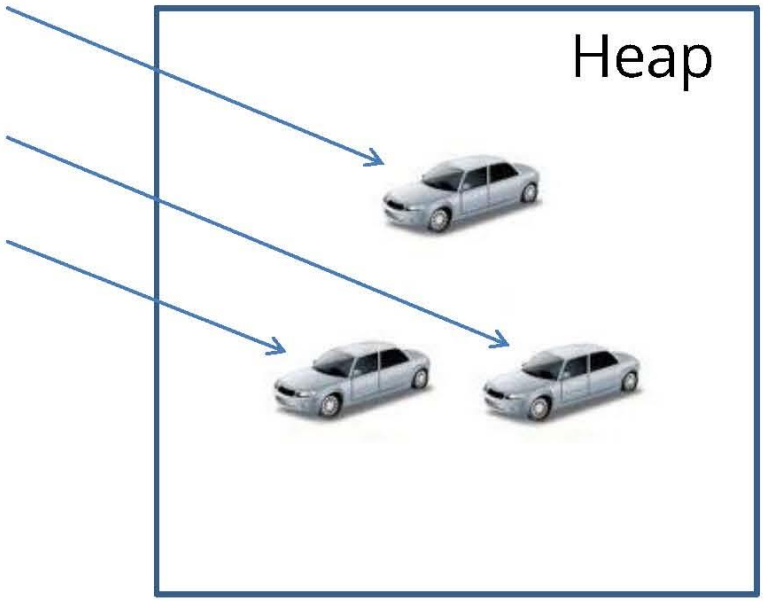


Creating Objects 2/3

```
1. public static void main(String[] arg) {

2.     Car car1 = new Car();
3.     Car car2 = new Car();
4.     Car car3 = new Car();

5. }
```



Creating Objects 3/3

```
1. public static void main(String[] arg) {

2.     Car car1 = new Car();
3.     Car car2 = new Car();
4.     Car car3 = new Car();

5.     // call method "getSpeed" of car1
6.     int speed1 = car1.getSpeed();
7.     // call method "getSpeed" of car2
8.     System.out.println(car2.getSpeed());
9. }
```

See objcreation

Encapsulation 2/3

- Encapsulation is one of the four fundamentals of the Object-Oriented Programming: ***Abstraction***, ***Encapsulation***, ***Inheritance*** and ***Polymorphism***.
- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties direct access to them.

objcreation - Encapsulate fields

Module contents

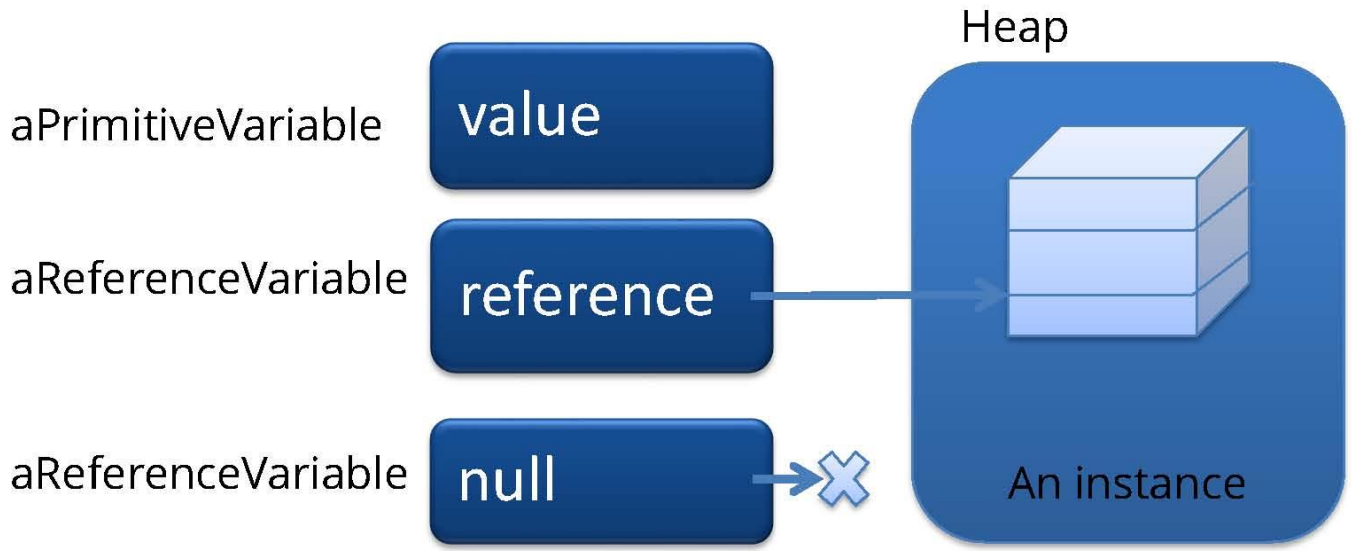
- Classes and Instances
  - Overview of class declarations. Class Fields and Methods.
  - Access modifiers
  - Encapsulation
  - Creating Objects
  - null literal
  - this Keyword

The null 1/3

- A reference variable refers to an object.
- When a reference variable does not have a value (it is not referencing an object) such a reference variable is said to have a null value.
- The null reference can always be assigned or cast to any reference type

The null 2/3

- null indicates that the object reference is not currently referring to an object



The null 3/3

1. String str1 = **null**; *// null can be assigned to String*
2. Car car1 = **null**; *// null can be assigned to Car*
3. **int** i = **null**; *// type mismatch : cannot convert from null to int*

4. String str2 = (String) **null**; *// null can be type cast to String*
5. Car car2 = (Car)**null**; *// null can be type cast to Car*

6. System.***out***.println(**null** == **null**); *// true*
7. System.***out***.println(car1 == **null**); *// true*
8. System.***out***.println(car1 == car2); *// true*

9. car1.getMaxSpeed();

- Exception in thread "main“ java.lang.NullPointerException

this keyword 1/2

- **this** is a reference to the *current object* — the object whose method or constructor is being called.

- You can refer to any member of the current object from within an instance method or a constructor by using **this**.