

Programming II

Lecture3 Objects & Classes

By the end of this lecture you will be able to...

- Motivate the use of classes and objects in programming.
- Write classes in Java
- Create objects and call methods on them
- Describe what member variables, methods and constructors are
- Give examples of overloading methods in Java
- Explain how to overload methods in Java
- Explain why overloading methods is useful
- Describe what the keywords public and private mean and their effect on where variables can be accessed
- Explain what getter and setters are and write them in your classes

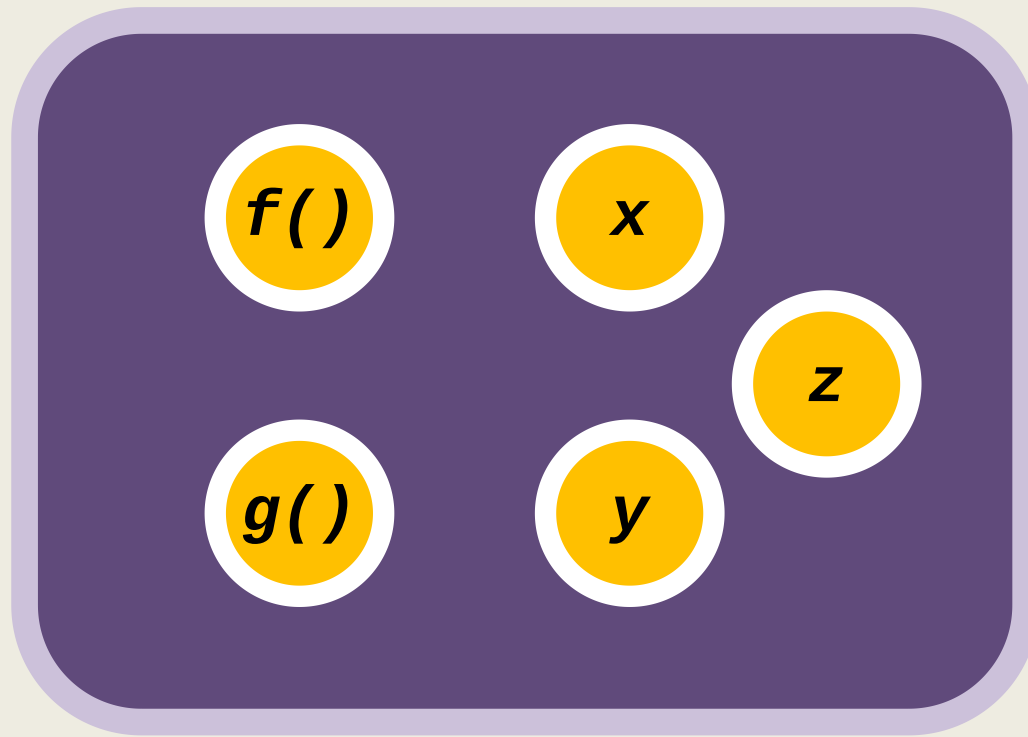


What is OOP?

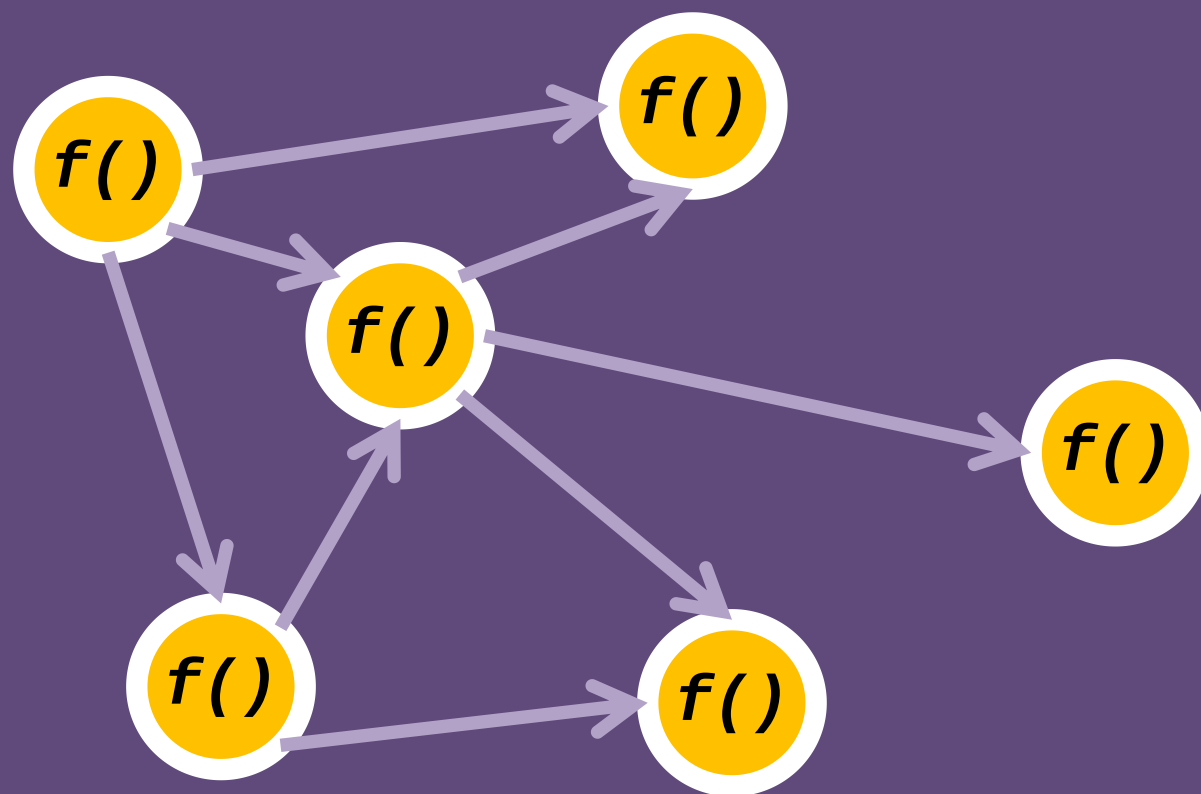
- Procedural programming is about writing a function that perform some calculations on the data.
- Object-oriented programming is about creating objects that contain related data and methods.

Procedural Programming

Program



Procedural Programming divided program into set of function

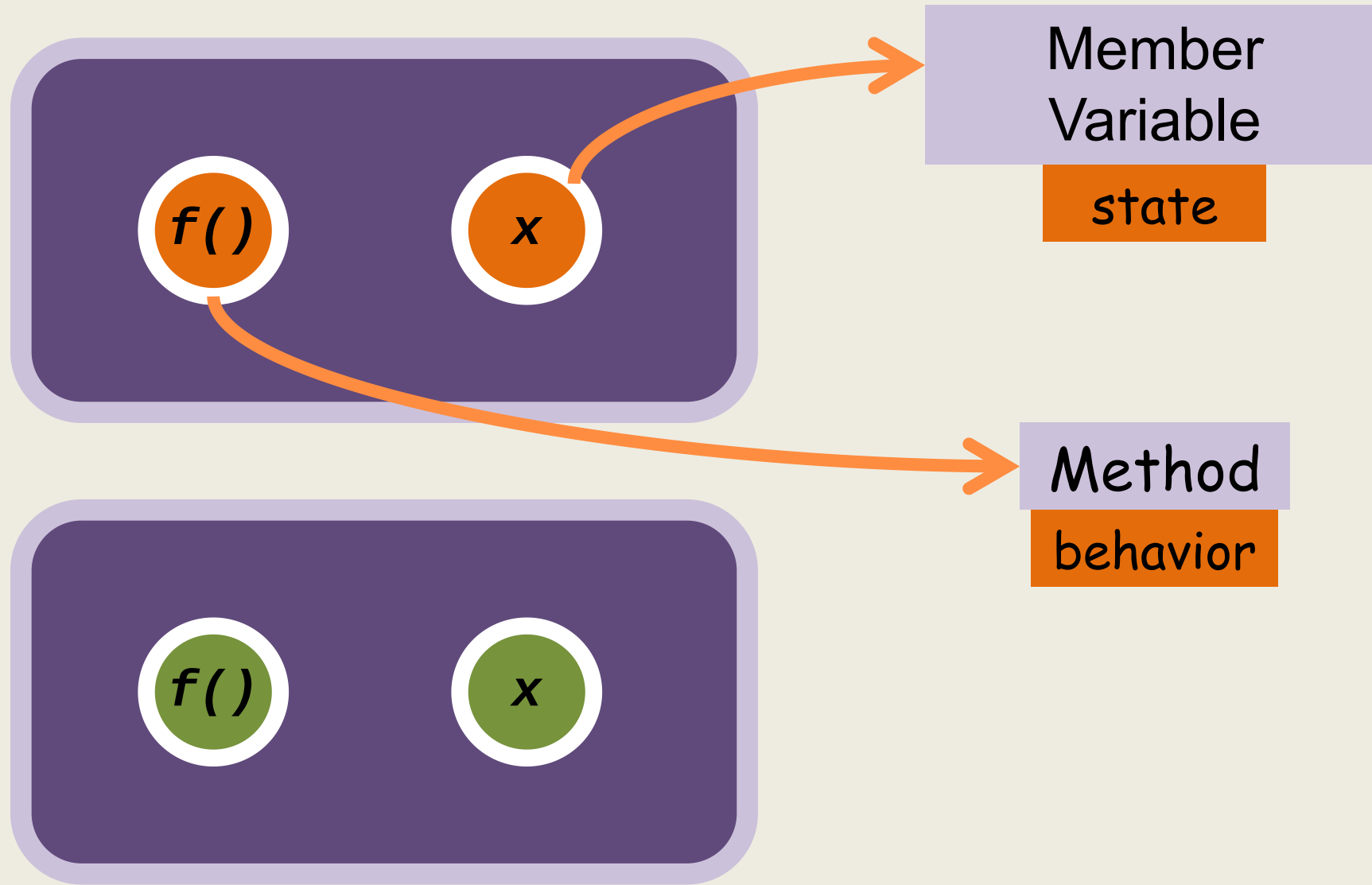




Object-oriented programming(OOP)

Object-oriented programming (OOP) involves programming using objects. An object represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a location can all be viewed as objects. An object has a unique identity, state, and behavior.

Object-oriented Programming



Car

make
model
color

start()
stop()
move()

Car

make
model
color

start()
stop()
move()

Give me a real
programming
example



> localStorage

< ▼ Storage {length: 0} ⓘ

length: 0

▼ __proto__: Storage

▶ clear: *f* clear()

▶ getItem: *f* getItem()

▶ key: *f* key()

length: (...)

▶ removeItem: *f* removeItem()

▶ setItem: *f* setItem()

▶ constructor: *f* Storage()

Symbol(Symbol.toStringTag): "Storage"

▶ get length: *f* length()

▶ __proto__: Object

>



Why OOP?

- How do we organize the information in a way such that our programs are:
 - Easy to write
 - Easy to maintain
 - Easy to debug



classes and objects



- What is a class?
 - A *class* is a **type** of data (**custom data types we get to define to the problem that we're trying to solve**).



- An object is one such **piece of data***

*With associated functionality



Defining a Class



UML notation of classes

The illustration of class templates and objects can be standardized using Unified Modeling Language (UML) notation. This notation is called a UML class diagram, or simply a class diagram. In the class diagram.

- The data field is denoted as

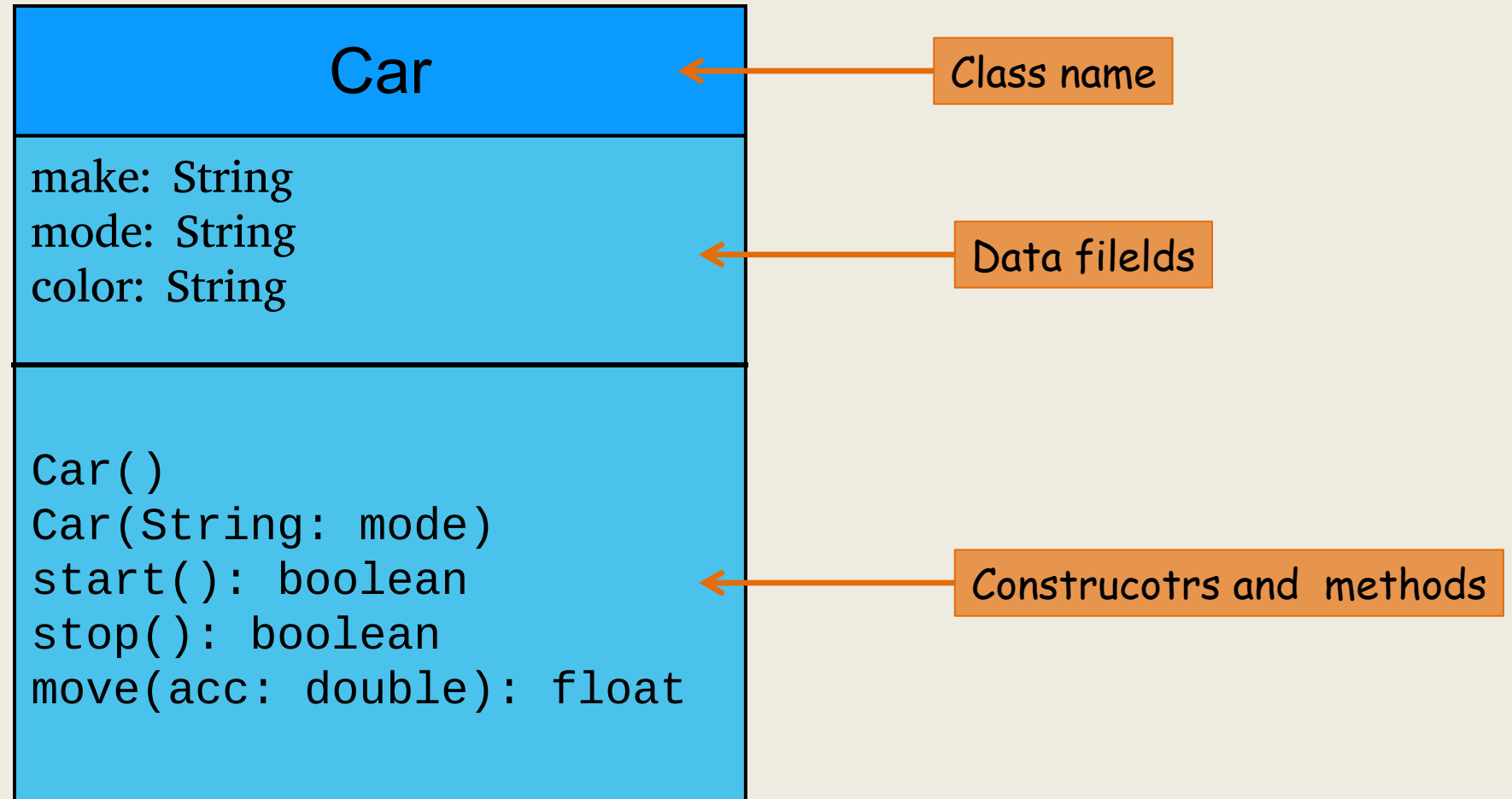
`dataFieldName: dataFieldType`

- The constructor is denoted as

`ClassName(parameterName: parameterType)`

- The method is denoted as

`methodName(parameterName: parameterType): returnType`





Map App



Map

Shape

Road

Location

... and plenty more object



Concept of Location
Latitude: 90.75
Longitude: -117.4

SimpleLocation

latitude: double
longitude: double

SimpleLocation()
showLocation(): void
distance(SimpleLocation): double



```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

Must be in file
SimpleLocation.java

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
    public void showLocation () {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }  
}
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;  
  
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }  
  
    public void showLocation () {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }  
}
```



```
public class SimpleLocation {
```

```
    public double latitude;  
    public double longitude;
```

Member/instance variables:
data the objects need to store

```
    public SimpleLocation(double lat, double lon) {
```

```
        this.latitude = lat;
```

```
        this.longitude = lon;
```

```
    }
```

```
    public void showLocation () {
```

```
        System.out.println("latitude: " + this.latitude);
```

```
        System.out.println("longitude: " + this.longitude);
```

```
    }
```

```
}
```

```
public class SimpleLocation {
```

```
    public double latitude;
```

```
    public double longitude;
```

Methods:

The things this class can do

```
    public SimpleLocation(double lat, double lon) {
```

```
        this.latitude = lat;
```

```
        this.longitude = lon;
```

```
    }
```

```
    public void showLocation () {
```

```
        System.out.println("latitude: " + this.latitude);
```

```
        System.out.println("longitude: " + this.longitude);
```

```
    }
```

```
}
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

Methods

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }  
  
    public void showLocation() {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }  
}
```

```
public class SimpleLocation {
```

```
    public double
```

```
    public double
```

Constructor:

Method to create a new object

```
    public SimpleLocation(double lat, double lon) {
```

```
        this.latitude = lat;
```

```
        this.longitude = lon;
```

```
    }
```

```
    public void showLocation () {
```

```
        System.out.println("latitude: " + this.latitude);
```

```
        System.out.println("longitude: " + this.longitude);
```

```
    }
```

```
}
```



```
public class SimpleLocation {
```

```
    public double
```

```
    public double
```

Constructor:
Method to create a new object

```
    public SimpleLocation(double lat, double lon) {
```

```
        this.latitude = lat;
```

```
        this.longitude = lon;
```

```
    }
```

```
    public void showLocation () {
```

```
        System.out.println("latitude: " + this.latitude);
```

```
        System.out.println("longitude: " + this.longitude);
```

```
    }
```

```
}
```



Constructors

- Constructors are a special kind of method. They have three peculiarities:
 - A constructor must have the same name as the class itself.
 - Constructors do not have a return type---not even **void**
 - Constructors are invoked using the **new** operators when an object is created. Constructors play the role of initializing objects.



Creating and using objects

In file
ClassDemo



```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
    }  
}
```

In file
ClassDemo

```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
    }  
}
```

Constructor:
Method to create a new object

```
public SimpleLocation(double lat, double lon) {  
    this.latitude = lat;  
    this.longitude = lon;  
}
```



Accessing an Object's Data and Methods

- In OOP terminology, an object's member refers to its data fields and methods. After an object is created, its data can be accessed and its methods can be invoked using the *dot operator*(.), also known as the object member access operator:
 - `objectRefVar.dataField` references a data field in the object
 - `objectRefVar.method`(arguments) invokes a method on the object.

```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
    }  
}
```


`ipm.showLocation();`



```
public void showLocation (){  
    System.out.println("latitude: " + this.latitude);  
    System.out.println("longitude: " + this.longitude);  
}
```



```
System.out.println(ipm.distance(borderGate));
```



```
public double distance (SimpleLocation others) {  
    /* calcualte distance between two points on the earth. */  
    // earth's radius in metres  
    final double R = 6371e3;  
    // change latitude and longitude to radians  
    double phi1 = this.latitude * Math.PI/180.0;  
    double phi2 = others.latitude * Math.PI/180.0;  
    double deltaPhi = (others.latitude - this.latitude) * Math.PI/180.0;  
    double deltaLamda = (others.longitude - this.longitude) * Math.PI/180.0;  
    // using haversine formula  
    double a = Math.sin(deltaPhi/2.0) * Math.sin(deltaPhi/2.0) +  
                Math.cos(phi1) * Math.cos(phi2) *  
                Math.sin(deltaLamda/2.0) * Math.sin(deltaLamda/2.0);  
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));  
    double d = R * c;  
  
    return d / 1000.0;  
}
```


```
System.out.println(ipm.distance(borderGate));
```



```
public double distance (SimpleLocation others) {  
    /* calculate distance between two points on the earth. */  
    // earth's radius in kilometres  
    final double R = 6371;  
    // change latitude and longitude to radians  
    double phi1 = this.latitude * Math.PI/180.0;  
    double phi2 = others.latitude * Math.PI/180.0;  
    double deltaPhi = (others.latitude - this.latitude) * Math.PI/180.0;  
    double deltaLamda = (others.longitude - this.longitude) * Math.PI/180.0;
```



```
System.out.println(ipm.distance(borderGate));
```



```
public double distance (SimpleLocation others) {  
    /* calculate distance between two points on the earth. */  
    // earth's radius in kilometres  
    final double R = 6371;  
    // change latitude and longitude to radians  
    double phi1 = this.latitude * Math.PI/180.0;  
    double phi2 = others.latitude * Math.PI/180.0;  
    double deltaPhi = (others.latitude - this.latitude) * Math.PI/180.0;  
    double deltaLamda = (others.longitude - this.longitude) * Math.PI/180.0;
```

"this" is the calling object



Overloading Methods

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

Must be in file
SimpleLocation.java

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
    public void showLocation () {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }
```

```
    public double distance (SimpleLocation others) {...}  
}
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

Constructor

```
    public void showLocation () {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }
```

```
    public double distance (SimpleLocation others) {...}  
}
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

Constructor without
parameters?

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
    public void showLocation () {  
        System.out.println("latitude: " + this.latitude);  
        System.out.println("longitude: " + this.longitude);  
    }
```

```
    public double distance (SimpleLocation others) {...}  
}
```



```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

```
    public SimpleLocation() {  
        this.latitude = 22.19;  
        this.longitude = 113.55;  
    }
```

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

```
    public SimpleLocation() {  
        this.latitude = 22.19;  
        this.longitude = 113.55;  
    }
```

Default constructor

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;
```

```
    public SimpleLocation() {  
        this.latitude = 22.19;  
        this.longitude = 113.55;  
    }
```

Overloading

```
    public SimpleLocation(double lat, double lon) {  
        this.latitude = lat;  
        this.longitude = lon;  
    }
```

```
public SimpleLocation(double lat, double lon) {...}
```

```
public void showLocation () {...}
```

```
public double distance (SimpleLocation others) {...}
```

```
public double distance (double otherLat, double otherLong) {...}
```



Why overloading ?

- Over loading methods can make programs clearer and more readable. Methods that perform the same function with different types of parameters should be given the same name.

```
public double distance (SimpleLocation others) {  
    // body not shown
```

```
}
```

CAUTION !

```
public int distance (SimpleLocation others){  
    // body not shown|
```

```
}
```



```
public double distance (SimpleLocation others) {  
    // body not shown
```

```
}
```

CAUTION!

```
public int distance (SimpleLocation others){  
    // body not shown
```

```
}
```

Return types is NOT part of the method signature.

Overloaded methods CAN have different return types.

Changing the return type is NOT ENOUGH for overloading.

!



Drawing Memory Models with Objects

 By the end of this section you will be able to...

- Draw memory models for reasoning about variable values for object type data
- Update memory models to trace the state of the variables in Java code

Primitive types vs. Object types

```
int var1 = 46;  
SimpleLocation ipm =  
    new SimpleLocation(lat: 22.19, lon: 113.55);  
SimpleLocation borderGate=  
    new SimpleLocation(lat: 22.21, lon: 113.55);
```

Primitive types vs. Object types

boolean, byte, short, int,
long, float, double, char

```
int var1 = 46;  
SimpleLocation ipm =  
    new SimpleLocation(lat: 22.19, lon: 113.55);  
SimpleLocation borderGate=  
    new SimpleLocation(lat: 22.21, lon: 113.55);
```

Primitive types vs. Object types

Arrays and Classes

```
int var1 = 46;  
SimpleLocation ipm =  
    new SimpleLocation(lat: 22.19, lon: 113.55);  
SimpleLocation borderGate=  
    new SimpleLocation(lat: 22.21, lon: 113.55);
```

```
int var1 = 46;
```



```
SimpleLocation ipm;
```

```
ipm = new SimpleLocation(lat: 22.19, lon: 113.55);
```

```
SimpleLocation borderGate = new SimpleLocation(lat: 22.21, lon: 113.55);
```

```
ipm.latitude = -27.08;
```

var1

46

```
int var1 = 46;  
SimpleLocation ipm; ←  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```

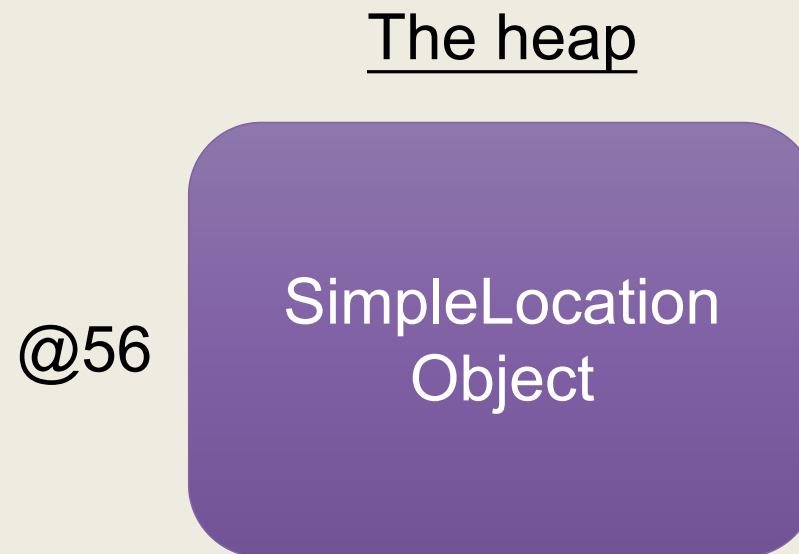
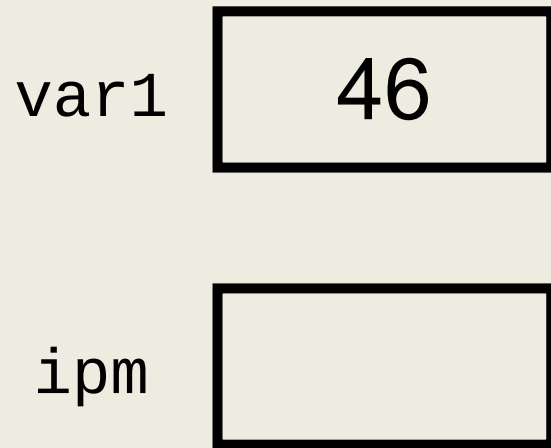
var1

46

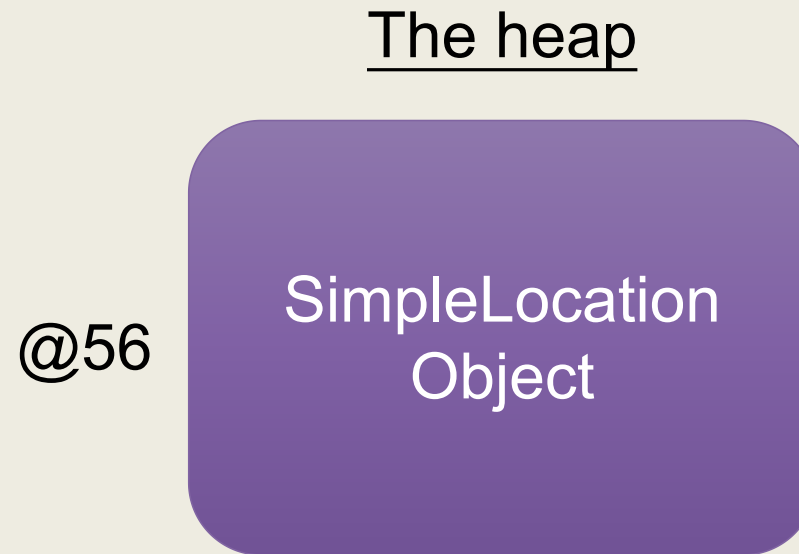
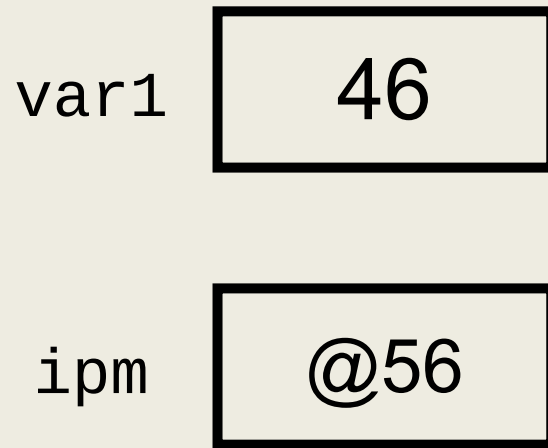
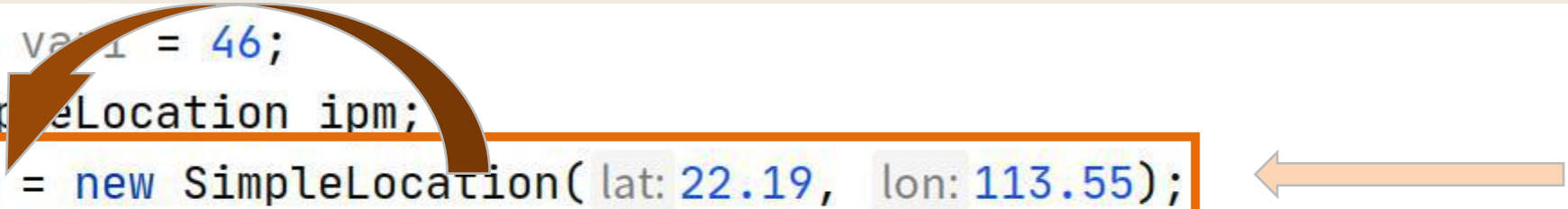
ipm

--

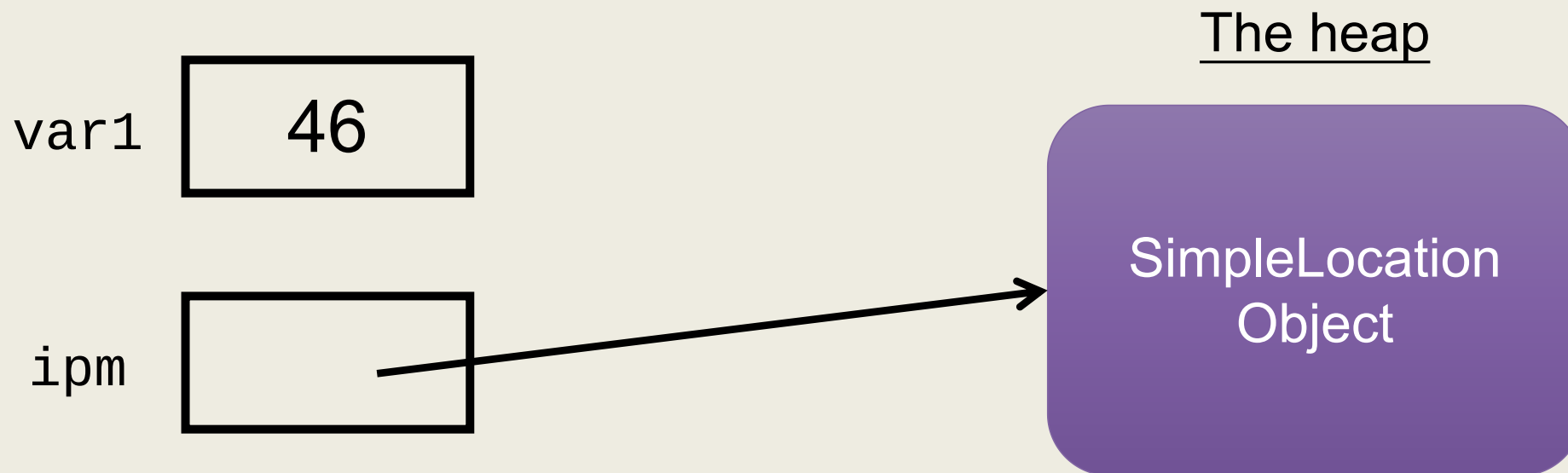
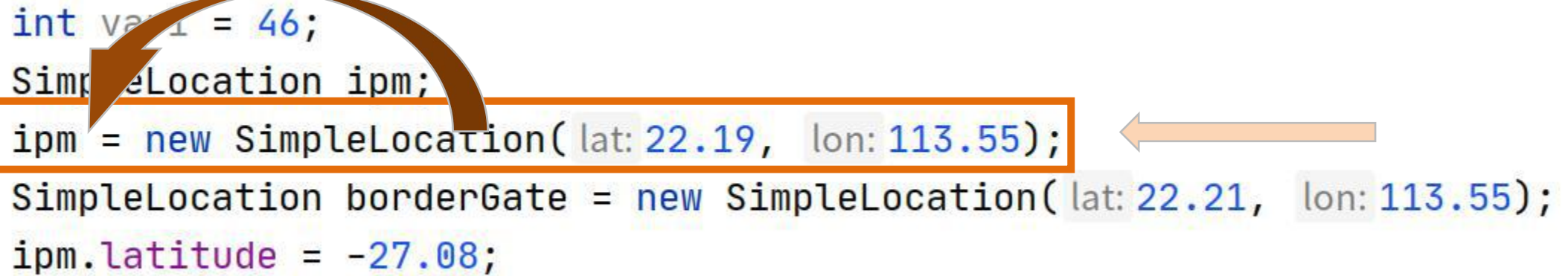
```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



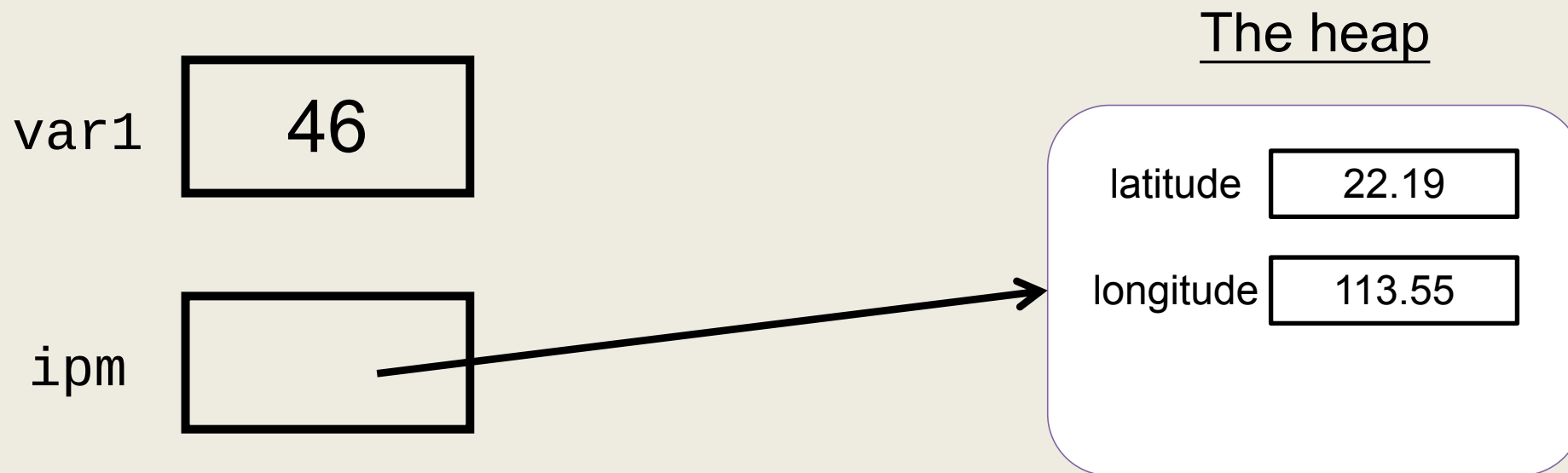


```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



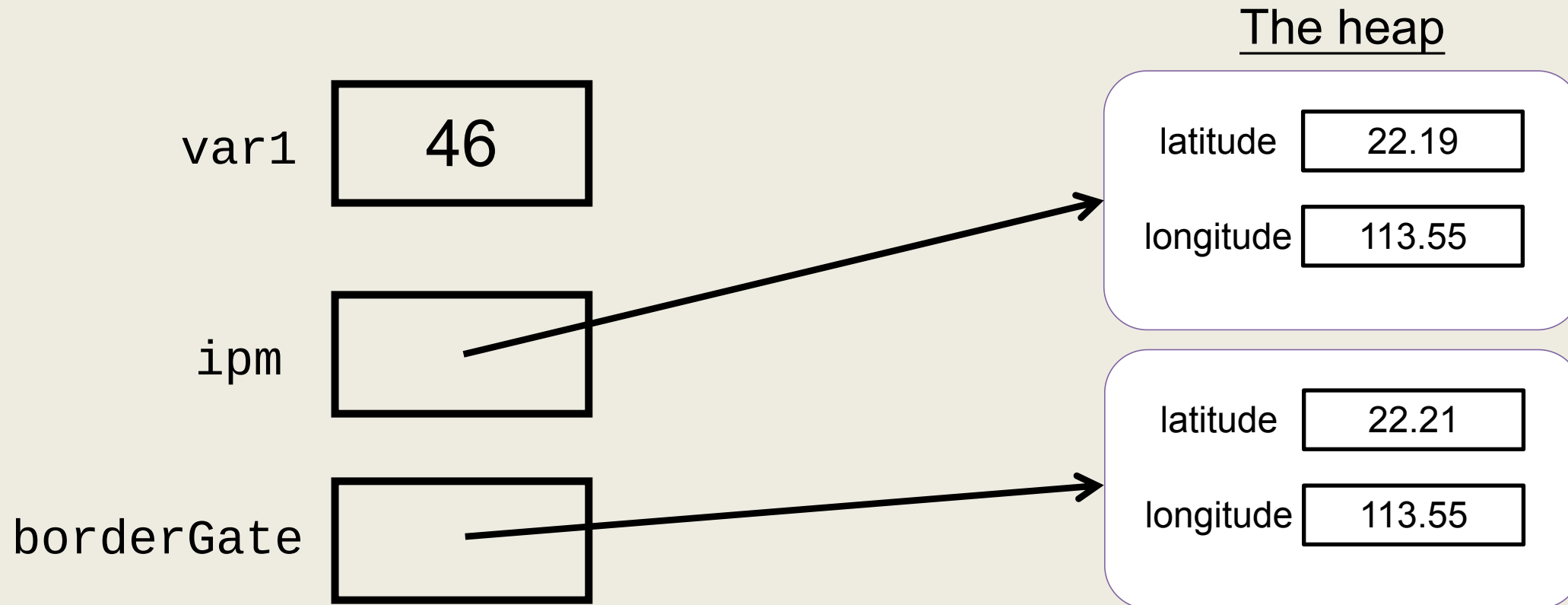
```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



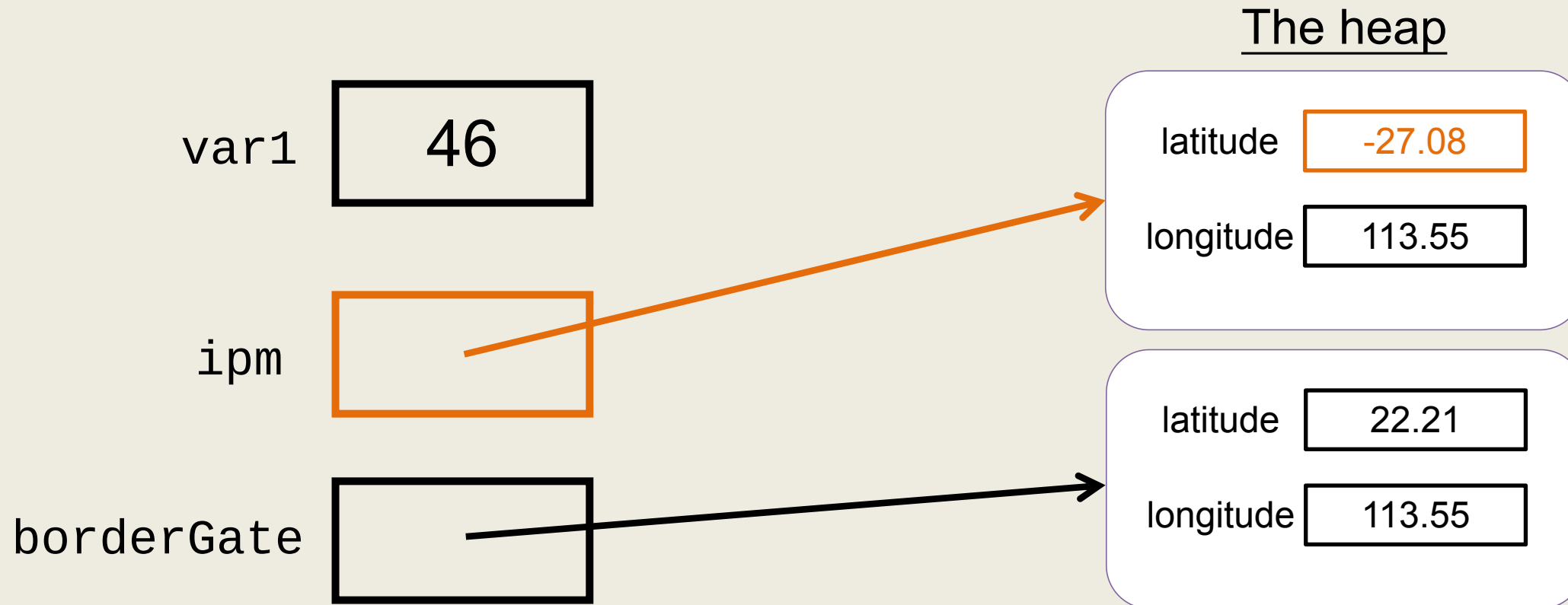
```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



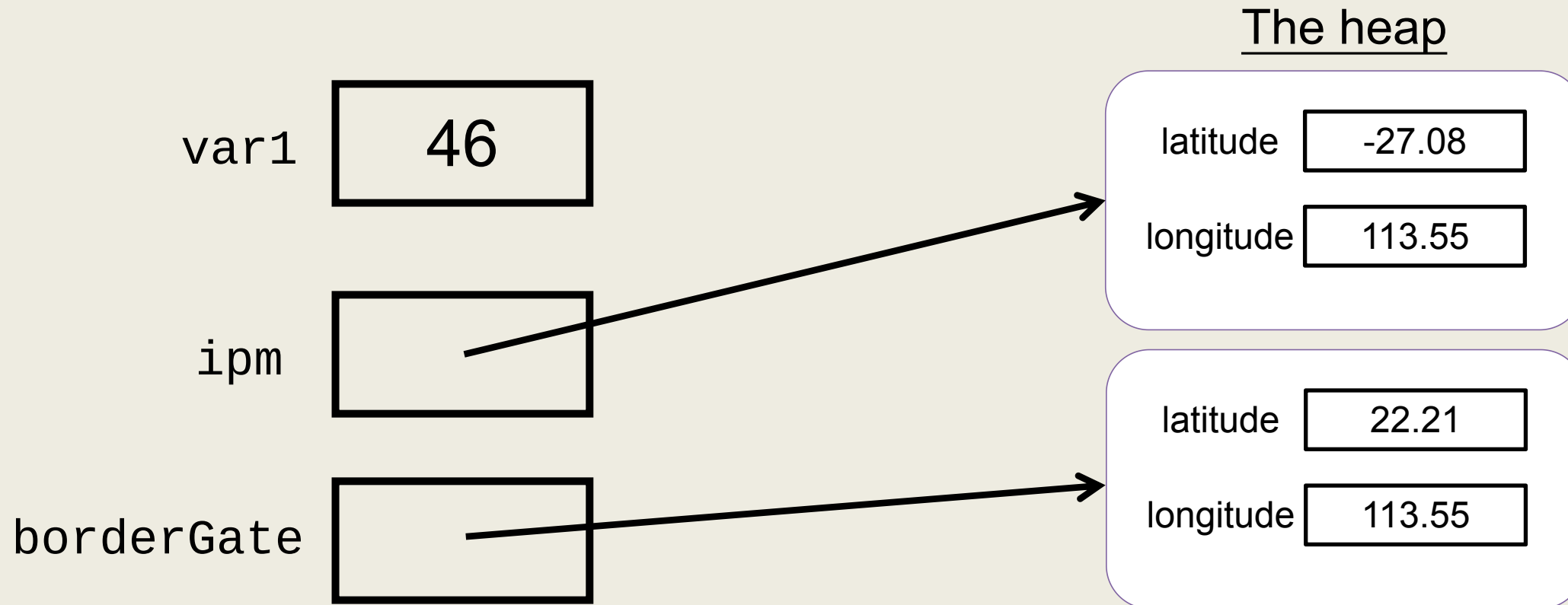
```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation( lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation( lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation(lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation(lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



```
int var1 = 46;  
SimpleLocation ipm;  
ipm = new SimpleLocation(lat: 22.19, lon: 113.55);  
SimpleLocation borderGate = new SimpleLocation(lat: 22.21, lon: 113.55);  
ipm.latitude = -27.08;
```



What does these codes print?

```
SimpleLocation loc1 = new SimpleLocation( lat: 29.9, lon: 122.4);  
SimpleLocation loc2 = new SimpleLocation( lat: 55.9, lon: -45.6);  
loc1 = loc2;  
loc1.latitude = -9.4;  
System.out.println(loc2.latitude + ", " + loc2.longitude);|
```




Public vs. Private



Encapsulation

- In Object-oriented programming we group related variables and functions that operate on them into objects and this is what we call **encapsulation**.
- Encapsulation in Java is mechanism of wrapping the data (variables) and code acting on the data (method) together as a single unit. In encapsulation, the member variable of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.
- To achieve encapsulation in Java-
 - Declare the variables of a class as private.
 - Provide public setter and getter methods to modify and view the variables values.



```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;  
}
```

public means can access from any class

```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.longitude = -24.03;  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
        System.out.println(ipm.distance(otherLat: 45.7, otherlong: 55.6));  
    }  
}
```

```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.longitude = -24.03;  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
        System.out.println(ipm.distance(otherLat: 45.7, otherlong: 55.6));  
    }  
}
```

allowed

```
public class SimpleLocation {  
    public double latitude;  
    public double longitude;  
  
    public SimpleLocation() {...}  
  
    public SimpleLocation(double lat, double lon) {...}  
  
    public void showLocation () {...}  
  
    public double distance (SimpleLocation others) {...}  
  
    public double distance (double otherLat, double otherLong) {...}
```



```
public class SimpleLocation {  
    private double latitude;  
    private double longitude;
```

private means can access only from SimpleLocation

```
    public SimpleLocation() {...}
```

```
    public SimpleLocation(double lat, double lon) {...}
```

```
    public void showLocation () {...}
```

```
    public double distance (SimpleLocation others) {...}
```

```
    public double distance (double otherLat, double otherLong) {...}
```



```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        ipm.longitude = -24.03;  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
        System.out.println(ipm.distance(otherLat: 45.7, otherlong: 55.6));  
    }  
}
```

ERROR
we are not in the
SimpleLacatio class





If the member variables are private,
nobody else can access them.

Giving the right
level of access

```
public class SimpleLocation {  
    private double latitude;  
    private double longitude;
```

```
    public double getLatitude(){  
        return this.latitude;  
    }
```



getter

```
public class ClassDemos {  
    public static void main(String[] args) {  
        SimpleLocation ipm =  
            new SimpleLocation(lat: 22.19, lon: 113.55);  
        SimpleLocation borderGate=  
            new SimpleLocation(lat: 22.21, lon: 113.55);  
  
        System.out.println(ipm.getLatitude());  
  
        ipm.showLocation();  
        System.out.println(ipm.distance(borderGate));  
        System.out.println(ipm.distance(otherLat: 45.7, otherlong: 55.6));  
    }  
}
```



Allowed

```
public class SimpleLocation {  
    private double latitude;  
    private double longitude;
```

```
    public double getLatitude(){  
        return this.latitude;  
    }
```



Can the user
change the
value?


getter

```
public class SimpleLocation {  
    private double latitude;  
    private double longitude;  
  
    public double getLatitude(){  
        return this.latitude;  
    }  
}
```


```
public void setLatitude(double lat){  
    this.latitude = lat;  
}
```



setter



Why didn't we just make
that member variable
public?



Why didn't we just make
that member variable
public?

Getters and Setters
give us more control



```
public void setLatitude(double lat){  
    if (lat < -180 || lat > 180){  
        System.out.println("Illegal value for latitude");  
    }  
    else {  
        this.latitude = lat;  
    }  
}
```




Using Classes From the Java Library



The Date Class

The + sign indicates
public modifier

java.util.Random



```
+Random()  
+Random(seed: long)  
+nextInt(): int  
+nextInt(n: int): int  
+nextLong(): long  
+nextDouble(): double  
+nextFloat(): float  
+nextBoolean(): boolean
```

```
import java.util.Random;
```

```
Random generator1 = new Random( seed: 3);  
System.out.print("From generator1: ");  
for (int i = 0; i < 10; i++){  
    System.out.print(generator1.nextInt( bound: 1000) + " ");  
}
```

From generator1: 734 660 210 581 128 202 549 564 459 961

```
System.out.println(Math.PI);
```

```
System.out.println(Math.cos(3.1415));
```

3.141592653589793

-0.99999999957076562


```
System.out.println(Math.PI);
```

```
System.out.println(Math.cos(3.1415));
```

3.141592653589793

-0.99999999957076562

Where is the object?



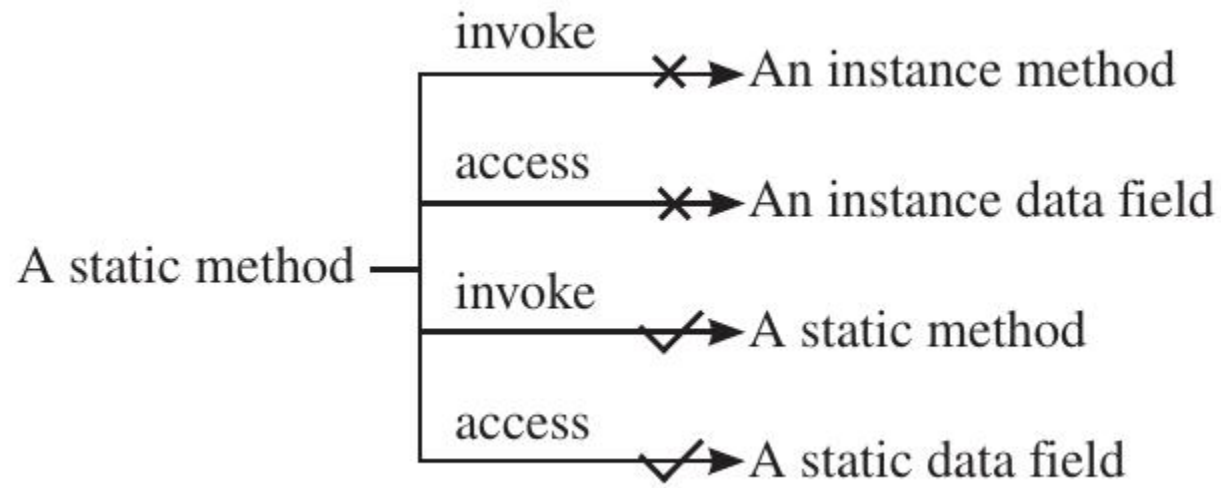
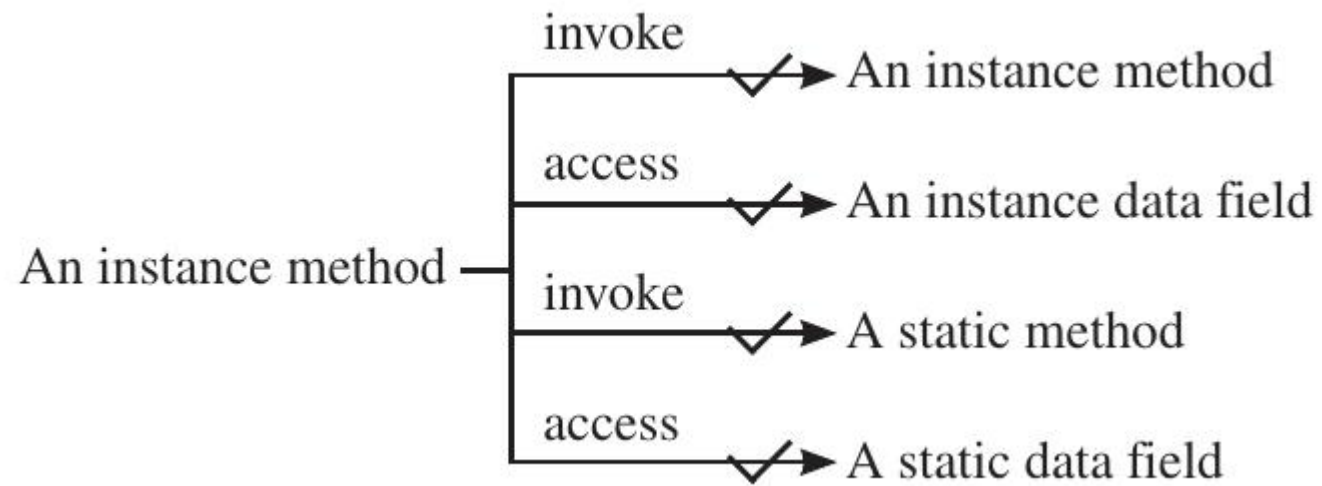
Static Variables, Constants and Methods



Static modifier

- An *instance variable* is tied to a specific instance of the class; it is not shared among objects of the same class.
- If you want all the instances of a class to share data, use static variables, also known as class variables. Static variables store values for the variables in a common memory location. Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected.
- Java supports static methods as well as static variables. Static methods can be called without creating an instance of the class.

DEMO





Immutable Objects and Classes



immutable objects and classes

- Normally, you create an object and allow its contents to be changed later. However, occasionally it is desirable to create an object whose contents cannot be changed once the object has been created. We call such an object as immutable object and its class as immutable class.
- For a class to be immutable, it must meet the following requirements:
 - All data fields must be private.
 - There can't be any mutator methods for data fields.
 - No accessor methods can return a reference to a data field that is mutable.



Benefits of immutable class

1. Simplicity

An immutable class can just be in one externally visible state. On the contrary, a mutable object can have a lot of states. Each mutation of the instance variable takes the object to another state.

2. Thread safety

They do not require synchronization. Since there is no way the state of an immutable object can change, there is no possibility of one thread observing the effect of another thread.

3. Enables reuse

Immutable objects encourage to cache or store the frequently used instances rather than creating one each time. This is because two immutable instances with the same properties/values are equal.