Lecture 1-2
Classes and Objects in Java

Department of Computer Science
Hofstra University

Lecture Goals

- Write classes, create objects, and call methods on them.
- Describe what member variables, methods and constructors are.
- Describe what the keywords public and private mean and their effect on where variables can be accessed
- Explain what getters and setters are and write them in your classes
- Explain how to overload methods in Java and why overloading methods is useful
- Draw memory models with variable scope for reasoning about variable values for object type data.

Reasons to Choose Java

- Promise of portability
  - write-once/run-anywhere
- Efficient memory management
  - garbage collection
- Powerful object-oriented programming
  - Inheritance and Polymorphism

Write Once and Run Anywhere

1. Write source code - HelloWorld.java

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

2. Compile source code - javac HelloWorld.java

Compiler

Java Bytecode

Obtain bytecode - HelloWorld.class

2. Run in JVM - java HelloWorld

Java Virtual Machine

Java API

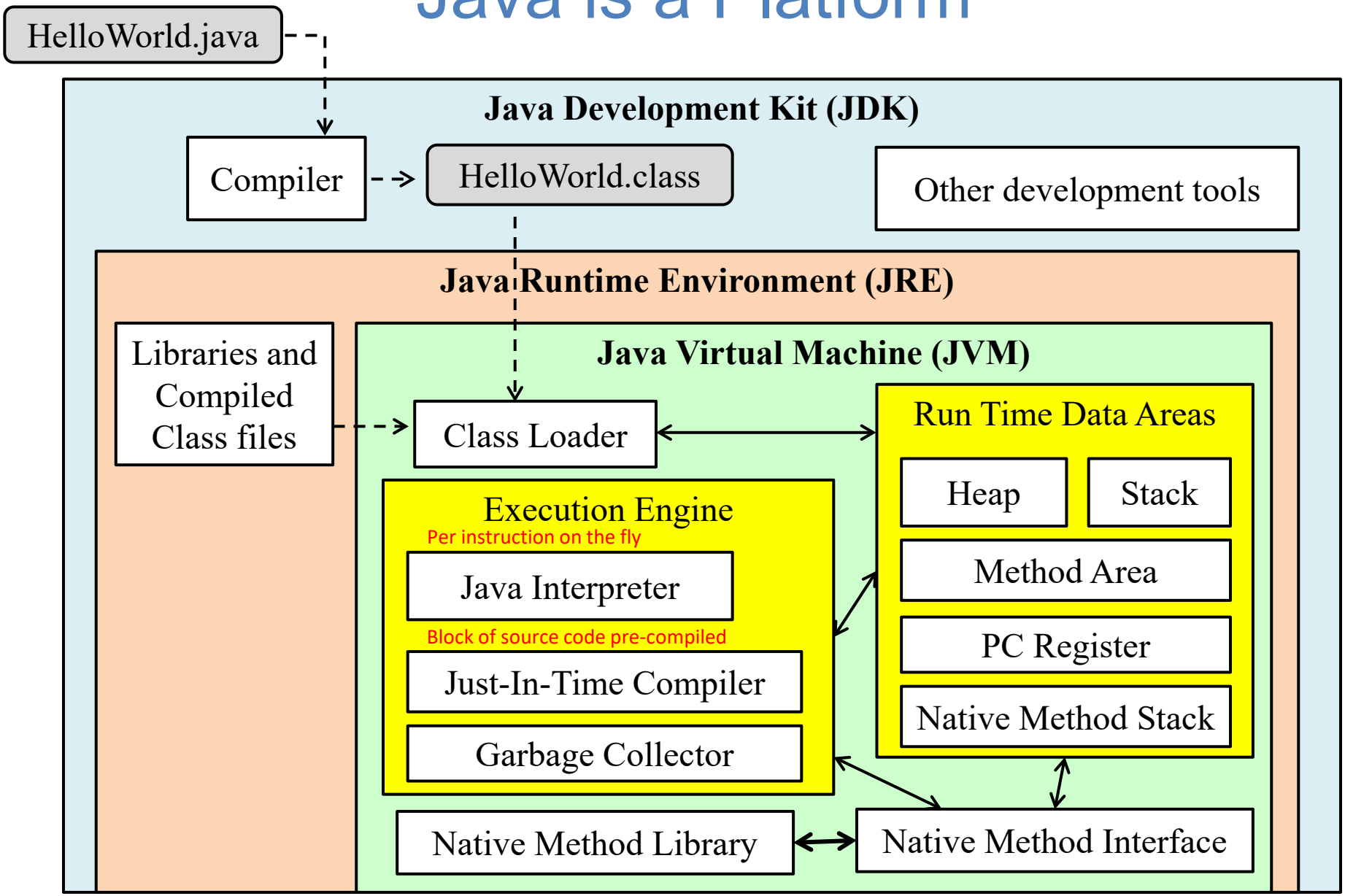
Native Machine Code

Operating Systems

Hardware

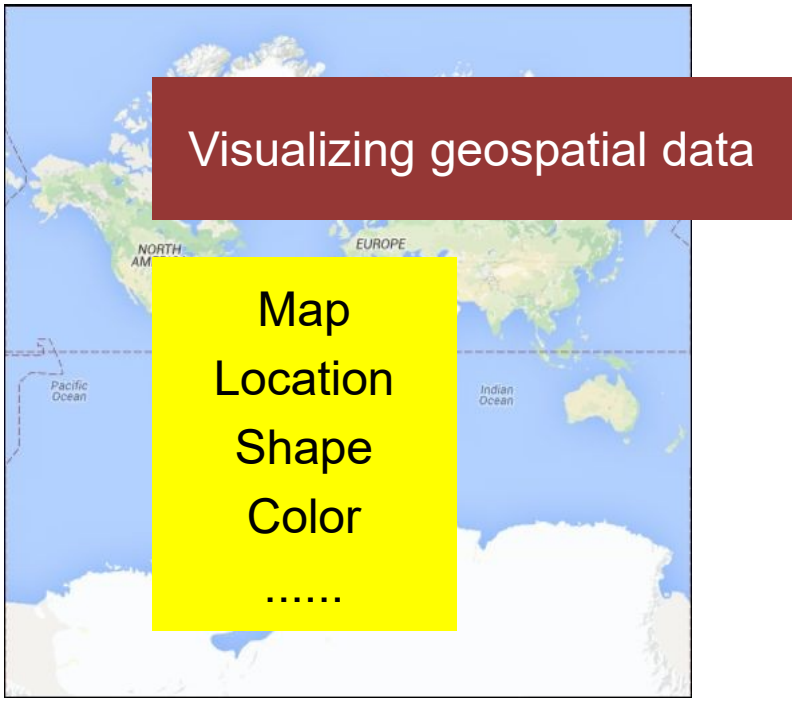
```
Jianchen$ java HelloWorld
Hello World
```

Java is a Platform



Object Oriented Programming (OOP)

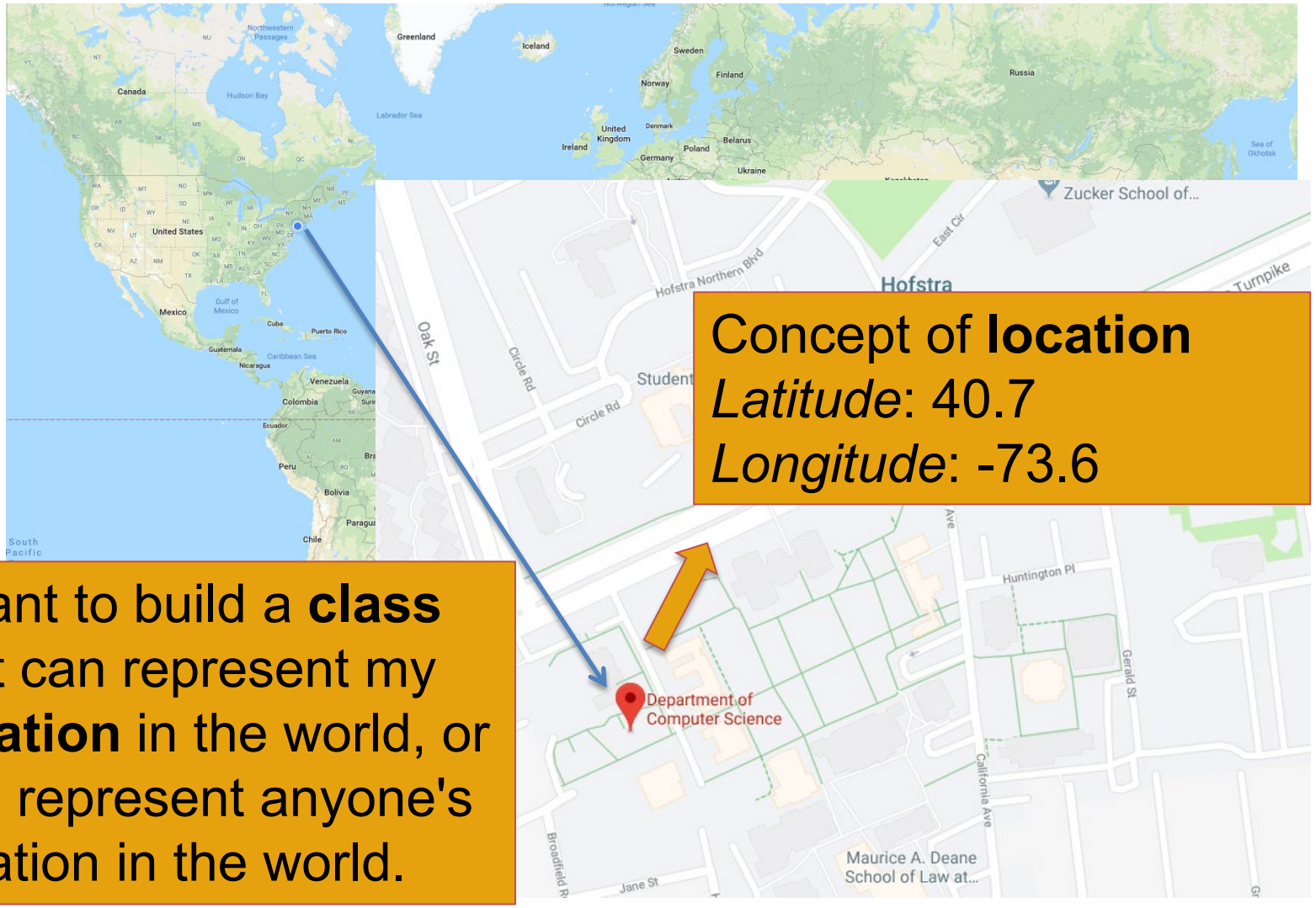
- *Computer science* -- is the science of using and processing large amounts of information to automate useful tasks and learn about the world around us using a computer.
- *OOP* -- organizes the information based on real-world objects such that program can be:
  - easy to match the problem
  - easy to write
  - easy to maintain
  - easy to debug



Definitions of Class and Object

- A *class* is a **type** of data
  - a template defined by the programmer
  - like a factory and can produce pieces of data with the template
- An *object* is one **such piece of data**
  - made out of the factory
  - with associated functionality
- A class can be used to produce multiple objects
- Each individual object can be customized and changed without affecting others

An Example of Class and Object



Defining a Class

Must be in file `Location.java`

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

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

    public double distance(Location other) {
        // body not shown
    }
}
```

Member variables:
data the objects need to store

Constructor:
Method to create a new object

Methods:
The things this class can do

Creating and Using Objects

In file `LocationTester.java`

```
public class LocationTester
{
    public static void main(String[] args)
    {
        Location hof = new Location(40.7,  -73.6);
    Location oxford = new Location(51.7,  -1.2);
    System.out.println(hof.distance(oxford));
    }
}
```

In file `Location.java`

```
public class Location
{
  public double latitude;
  public double longitude;
  public Location(double lat, double lon)
  {
        this.latitude = lat;
        this.longitude = lon;
    }
    public double distance(Location other) {
        // body not shown
    }
}
```

"`this`" is the calling object

Creating and Using Objects (Contd.)

In file `LocationTester.java`

```
public class LocationTester
{
    public static void main(String[] args)
    {
        Location hof = new Location(40.7,  -73.6);
    Location oxford = new Location(51.7,  -1.2);
    System.out.println(hof.distance(oxford));
    }
}
```

```
$ javac *.java
$ java LocationTester
3397.26
```

In file `Location.java`

```
public class Location
{
  public double latitude;
  public double longitude;
  public Location(double lat, double lon)
  {
                this.latitude = lat;
                this.longitude = lon;
    }
  public double distance(Location other) {
                return getDist(this.latitude, this.longitude,
        other.latitude, other.longitude);
  }
}
```

"`this`" is the calling object *hof*

The Main Method in Java

- Java begins execution with the first line of a "main" method

  public static void main(String[] args)

- This method can be defined in any class, usually ***public***.
- When a class has a main method, it can be "run" and all other is only one class with m
- The keyword ***st*** iable, simply means th the class, but not for **eral method**.
- There is no "call e methods from m nce methods on thos ods directly.

```
public class Location
{
  public double latitude;
  public double longitude;
  public Location(double lat, double lon)
  {
                this.latitude = lat;
                this.longitude = lon;
    }
   public double distance(Location other) {
                return getDist(this.latitude, this.longitude,
          other.latitude, other.longitude);
  }
  public static void main(String[] args) {
                Location hof = new Location(40.7,  -73.6);
                Location oxford = new Location(51.7,   1.2);
                this.distance(hof);
                hof.distance(oxford);
  }
}
```

✗

✓

Overloading Methods

In file
`Location.java`

```
public class Location
{
   public double latitude;
   public double longitude;
   public Location(double lat, double lon)
   {
            this.latitude = lat;
            this.longitude = lon;
      }
      public double distance(Location other) {
            // body not shown
      }
}
```

What if the user wants to create Location objects without passing in any parameters?

Overloading Methods (Contd.)

```
public class Location
{
        public double latitude;
        public double longitude;
        public Location() {
                this.latitude = 40.7;
                this.longitude = -73.6;
        }
        public Location(double lat, double lon) {
                this.latitude = lat;
                                                ) {
        }
}
```

In file `Location.java`

Constructor without parameters

Default constructor

Overloading

Parameter constructor

Overloading Methods (Contd.)

In file `Location.java`

```
public class Location
{
        // Code omitted here
        public double distance(Location other)
        {
                // body not shown
        }
        public double distance(double otherLat, double otherLon)      {
                // body not shown
        }
}
```

What is the advantage? We don't have to create and remember different names for functions doing the same thing. For example, in our code, if overloading was not supported by Java, we would have to create method names like `distance1` and `distance2`.

A Real-world Example of Overloading

- ArrayList in Java API: overloaded constructors and `add` method

Constructors

Constructor and Description
`ArrayList()` Constructs an empty list with an initial capacity of ten.
`ArrayList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the
`ArrayList(int initialCapacity)` Constructs an empty list with the specified initial capacity.

Methods

Modifier and Type	Method and Description
`boolean`	**`add(E e)`** Appends the specified element to the end of this list.
`void`	**`add(int index, E element)`** Inserts the specified element at the specified position in this list.

CAUTION

```
public class Location
{
        // Code omitted here
        public double distance(Location other)
        {
                // body not shown
        }
        public int distance(Location other)
        {
                // body not shown
        }
}
```

In file `Location.java`

Parameter must be different

At compile time, the compiler decides which version of the overloaded method you're actually trying to call by using the parameter list. It can't do that by using the return type alone.

Public vs. Private: Protect Data and Method

In file
`Location.java`

```
public class Location
{
        public double latitude;
        public double longitude;
        public Location(double lat, double lon) {
                this.latitude = lat;
                this.longitude = lon;
        }
        public double distance(Location other) {
                // body not shown
        }
}
```

public means can access from any class

In file
`LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
                Location hof = new Location(40.7,  -73.6);
        Location oxford = new Location(51.7,  -1.2);
                hof.latitude = 35.2;
        System.out.println(hof.distance(oxford));
        }
}
```

allowed

Public vs. Private: Protect Data and Method

In file `Location.java`

```
public class Location
{
        private double latitude;
        private double longitude;
        public Location(double lat, double lon) {
                this.latitude = lat;
                this.longitude = lon;
        }
        public double distance(Location other) {
                // body not shown
        }
}
```

private means can access only from `Location`

allowed

In file `LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
                Location hof = new Location(40.7,  -73.6);
                Location oxford = new Location(51.7,  -1.2);
                hof.latitude = 35.2;
                System.out.println(hof.distance(oxford));
        }
}
```

ERROR

Basic Class Design Rules

Rule of thumb: Make member variables private (and methods either public or private)

Methods

Private: helper methods

Public: for world use

Members

Private: use getters and setters

giving right level of access

An Example of Getter

In file
`Location.java`

```
public class Location
{
        private double latitude;
        private double longitude;
        // code omitted here
        public double getLatitude()
        {
                return this.latitude;
        }
}
```

getter

Can the user change the value ?

In file
`LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
                Location hof = new Location(40.7,  -73.6);
                System.out.println(hof.latitude);
                System.out.println(hof.getLatitude());
        }
}
```

ERROR

allowed

An Example of Setter

In file
`Location.java`

```
public class Location
{
        private double latitude;
        private double longitude;
        // code omitted here
        public void setLatitude(double lat)
        {
                this.latitude = lat;
        }
}
```

setter

why don't we just make that member variable public? If we're exposing the ability to change and read it?

In file
`LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
                Location hof = new Location(40.7,  -73.6);
                hof.latitude = 35.2;
                hof.setLatitude(35.2);
        }
}
```

ERROR

allowed

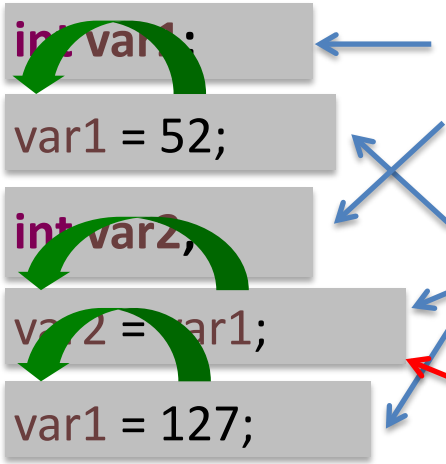
Another Example of Setter

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

getters and setters give us more control

Trace Your Code: Drawing Memory Model

what does this code print?



Variable declaration: draw a box and label it with the variable's name

Variable assignment: put the value of the right hand side into the box for the variable on the left hand side

NOT connected and just copy the value

```
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```


```
$ var1 is 127, var2 is 52
```

Primitive type data: int,double, float, short, long, char, boolean, byte

Drawing Memory Model with Objects

In file
`Location.java`

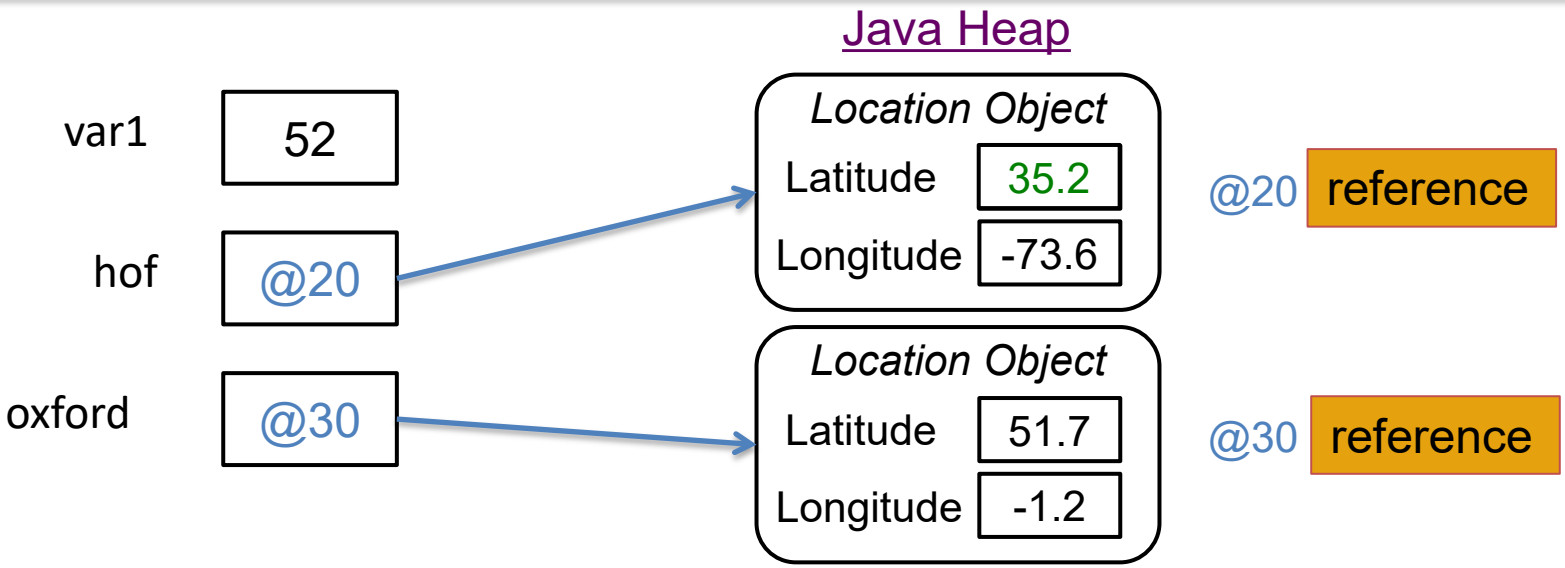
```
public class Location
{   private double latitude;
        // Code omitted here
        public static void main(String[] args)
        {
            int var1 = 52;
            Location hof;
            hof = new Location(40.7, -73.6);
            Location oxford = new Location(51.7 , -1.2);
            hof.latitude = 35.2;
```

variable declaration and same as primitives

assignment statement

memory reference

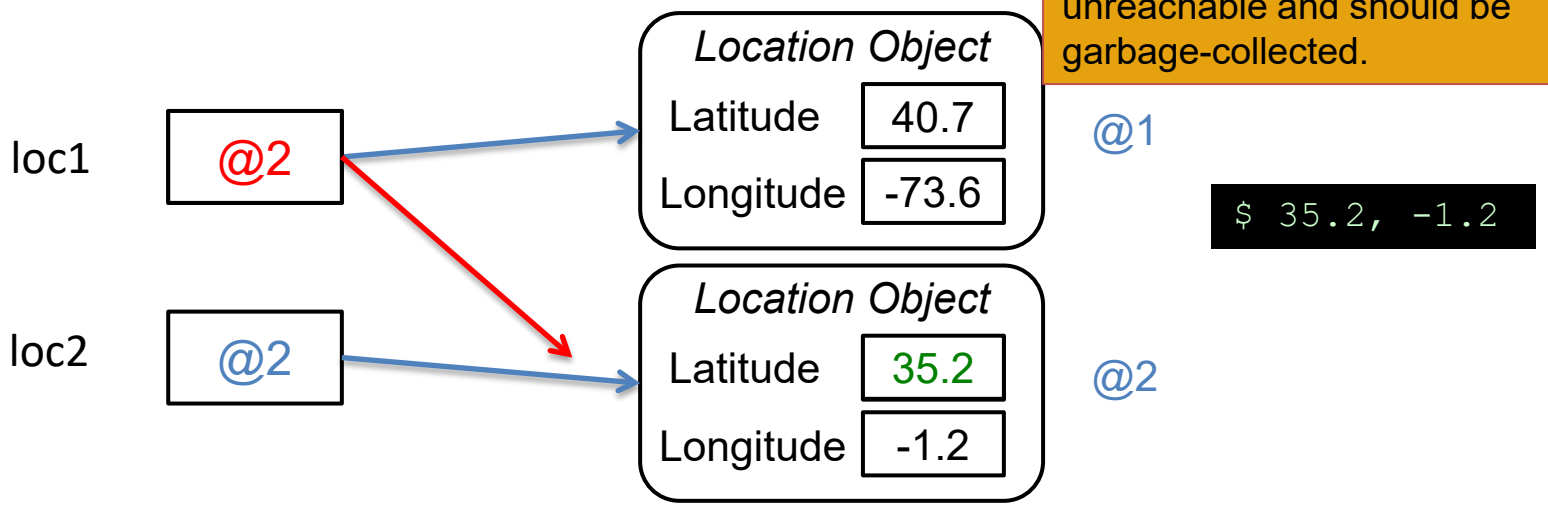
// in main method and can access private var



More Examples

```
public class Location
{
        // Code omitted here
        public static void main(String[] args)
        {
                Location loc1 = new Location(40.7, -73.6);
                Location loc2 = new Location(51.7 , -1.2);
                loc1 = loc2;
                loc1.latitude = 35.2;
                System.out.println(loc2.latitute + ", " + loc2.longitude);
```

After assignment loc1 = loc2, the Object Location(40.7, -73.6) is unreachable and should be garbage-collected.



Reason Your Code with Scope

In file `Location.java`

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

Member variables are declared outside any method

Parameters behave like local variables

In file `LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
            Location hof = new Location(40.7,  -73.6
            hof.latitude = 2.5;
        }
}
```

Local variables are declared inside a method

ERROR. Variable not defined here

The scope of a variable is the area where it is defined to have a value

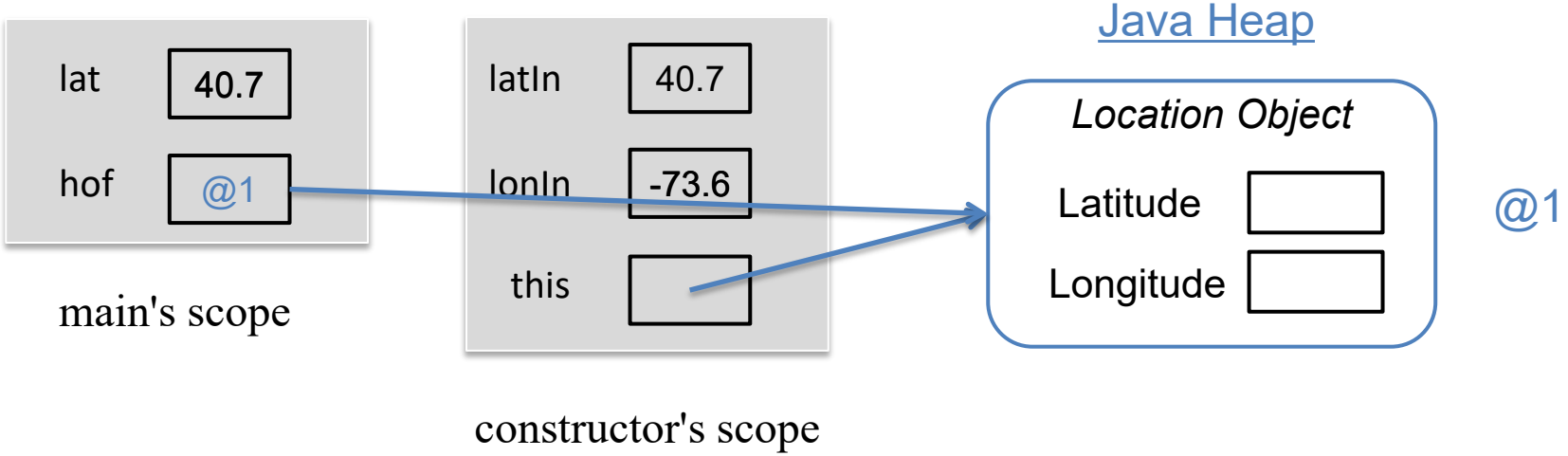
An Example

In file `Location.java`

```
public class Location
{
        public double latitude;
        public double longitude;
        public Location(double latIn, double lonIn) {
                this.latitude = latIn;
                this.longitude = lonIn;
        }
}
```

In file `LocationTester.java`

```
public class LocationTester
{
        public static void main(String[] args)
        {
                double lat = 40.7;
                Location hof = new Location(lat,  -73.6);
        }
}
```



An Example (Contd.)

In file Location.java

```
public class Location
{
        public double latitude;
        public double longitude;
        public Location(double latIn, double lonIn) {
                latitude = latIn;
                longitude = lonIn;
        }
}
```

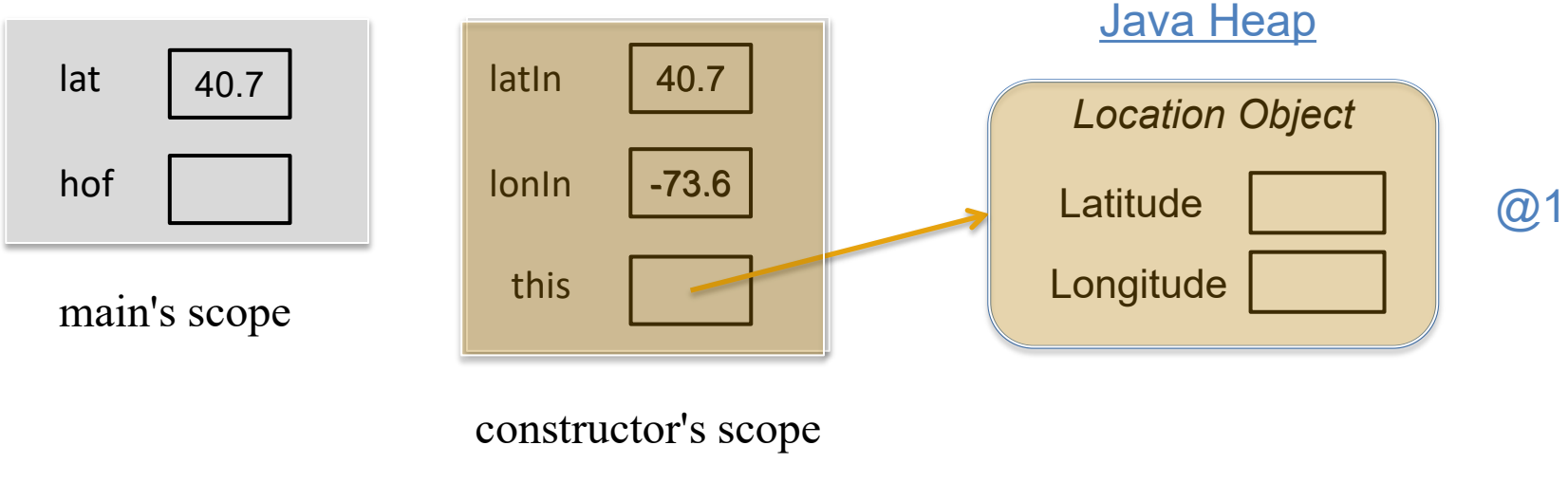
In file LocationTester.java

```
public class LocationTester
{
        public static void main(String[] args)
        {
                double lat = 40.7;
                Location hof = new Location(lat,  -73.6);
        }
```

Looks for latitude in the constructor's local scope

Doesn't find it, so looks in calling object scope

this is optional



Another Example

```
public class ArrayLocation
{
        private double coords[];
        public ArrayLocation(double[] coords) {
                this.coords = coords;
        }
        public static void main(String[] args)
        {
                double[] coords = {5.0, 0.0};
                ArrayLocation hof = new ArrayLocation(coords);
                coords[0] = 40.7;
                coords[1] = -73.6;
                System.out.println(hof.coords[0]);
        }
}
```

In file `ArrayLocation.java`

```
$ 40.7
```