Modeling the World

CS 151-06, Object-Oriented Design, Spring 2023

In this week

- 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
- Objects stored in memory
- Function Overloading
- Four Pillars of OOP
- Memory Modelling
- Java Heap vs Stack

Classes and Objects

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

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

What is OOP?

Object-oriented programming is a programming paradigm built on the **concept of objects**. In Other Words, it is an approach to problem-solving where all computations are carried out using objects.

- Program is divided into small parts called objects.
- Object-oriented programming follows a bottom-up approach.
- Have access specifiers like private, public, protected, etc.
- Adding new data and functions is easy.
- Provides data hiding so it is more secure than procedural programming.
- Overloading is possible in object-oriented programming.
- Data is more important than function.
- Provides the ability to simulate real-world
- Examples: C++, Java, Python, C#, JavaScript, Ruby, PHP, VB.NET

Classes and Objects

TERMINOLOGIES

Class

A class is a group of objects that share common properties and behavior. It is a **blueprint** or **template** from which objects are created.
A class is a type of data.

Object

Object is any real-world entity that can have some **characteristics**, or which can **perform some tasks**. It is also called the **instance** of a class.
An object is one such piece of data (with associated functionality)

Class

- Classes are defined using the `class`
- <u>non-primitive</u> or <u>user-defined</u> ***data type in Java***
- group of objects that share common properties and behavior
- A class defines the shared characteristics like –
  - The set of attributes/properties
  - The set of behavior or methods or actions
  - How to construct an object
- It is a logical entity that does not occupy any space/memory. Memory is allocated when we create the objects of a class type.

Object

- In Java, an object is created using the keyword `new`
- An object is an identifiable entity with some characteristics, state and behavior
- Understanding the concept of objects is much easier when we consider real-life examples around us because an object is simply a real-world entity.

Constructor

- Constructors are special methods whose name is the same as the class name. The constructors serve the special purpose of initializing the objects.

- **Default constructor** - The default constructor is the constructor which doesn’t take any argument. It has no parameters.

- **Parameterized constructor** - The constructors that take some arguments are known as parameterized constructors.

- Each time an object is created using a **new** keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the **data members** of the same class.

Constructor

How Constructors are Different From Methods in Java?

- Constructors must have the **same name** as the class within which it is defined while it is not necessary for the method in Java.
- Constructors **do not return any type** while method(s) have the return type or void if does not return any value.
- Constructors are **called only once at the time of Object creation** while method(s) can be called any number of times.

- Java supports:
  - copy constructor
  - constructor chaining (this()/ super())

Java Destructor

- **there is no concept of destructor in Java**
- The **destructor** is the opposite of the constructor. The constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object.
- In place of the destructor, Java provides the garbage collector that works the same as the destructor. The garbage collector is a program (thread) that runs on the JVM. It automatically deletes the unused objects (objects that are no longer used) and free-up the memory. The programmer has no need to manage memory, manually. It can be error-prone, vulnerable, and may lead to a memory leak.

Association, Composition and Aggregation in Java

- https://www.geeksforgeeks.org/association-composition-aggregation-java/

Four Pillars of OOP

- **Abstraction**
  We try to obtain abstract view, model or structure of real-life problem, and reduce its unnecessary details. With definition of properties of problems, including the data which are affected and the operations which are identified, the model abstracted from problems can be a standard solution to this type of problems. It is an efficient way since there are nebulous real-life problems that have similar properties.
- **Encapsulation**
  Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible. (Data hiding: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies. e.g., "protected", "private" feature in Java)
- **Inheritance**
  The idea of inheritance is simple, a class is based on another class and uses data and implementation of the other class. And the purpose of inheritance is Code Reuse.
- **Polymorphism**
  Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. A point shape needs only two co-ordinates (assuming it's in a two-dimensional space of course). A circle needs a center and radius. A square or rectangle needs two co-ordinates for the top left and bottom right corners and (possibly) a rotation. An irregular polygon needs a series of lines.

Function Overloading

two or more methods may have the same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

- How to perform method overloading in Java?

1. Overloading by **changing the number of parameters**
2. Method Overloading by **changing the data type of parameters**

We **cannot** overload by return type

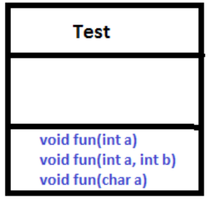
Function Overloading

- **Can we overload static methods?**
  Yes. Refer here.
- **Can we overload methods that differ only by static keyword?**
  No. We **cannot** overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same).
- **Can we overload main() in Java?**
  Like other static methods, we can overload main() in Java.
- **Does Java support Operator Overloading?**
  Unlike C++, Java doesn’t allow user-defined overloaded operators. Internally Java overloads operators, for example, + is overloaded for concatenation.

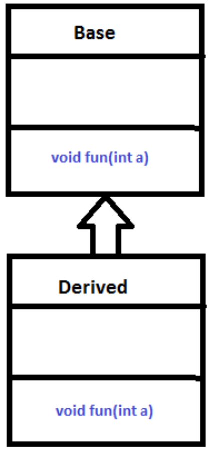
Overloading vs. Overriding

- Overloading is about same function have different signatures. Overriding is about same function, same signature but different classes connected through inheritance.

- Overloading is an example of compiler time polymorphism and overriding is an example of run time polymorphism.



Overloading



Overriding

Java memory and heap space

- To run an application in an optimal way, JVM divides memory into stack and heap memory. **Whenever we declare new variables and objects, call a new method, declare a *String,* or perform similar operations, JVM designates memory to these operations from either Stack Memory or Heap Space.**

Stack Memory

- **Stack Memory in Java is used for static memory allocation and the execution of a thread.** It contains primitive values that are specific to a method and references to objects referred from the method that are in a heap.
- Access to this memory is in Last-In-First-Out (LIFO) order. Whenever we call a new method, a new block is created on top of the stack which contains values specific to that method, like primitive variables and references to objects.
- When the method finishes execution, its corresponding stack frame is flushed, the flow goes back to the calling method, and space becomes available for the next method.

Key Features of Stack Memory

- Some other features of stack memory include:
- It grows and shrinks as new methods are called and returned, respectively.
- Variables inside the stack exist only as long as the method that created them is running.
- It's automatically allocated and deallocated when the method finishes execution.
- If this memory is full, Java throws *java.lang.StackOverFlowError*.
- Access to this memory is fast when compared to heap memory.
- This memory is threadsafe, as each thread operates in its own stack.

Heap Space in Java

Heap space is used for the dynamic memory allocation of Java objects and JRE classes at runtime. New objects are always created in heap space, and the references to these objects are stored in stack memory.

- These objects have global access and we can access them from anywhere in the application.
- We can break this memory model down into smaller parts, called generations, which are:
- Young Generation – this is where all new objects are allocated and aged. A minor Garbage collection occurs when this fills up.
- Old or Tenured Generation – this is where long surviving objects are stored. When objects are stored in the Young Generation, a threshold for the object's age is set, and when that threshold is reached, the object is moved to the old generation.
- Permanent Generation – this consists of JVM metadata for the runtime classes and application methods.

Key Features of Java Heap Memory
Some other features of heap space include:
It's accessed via complex memory management techniques that include the Young Generation, Old or Tenured Generation, and Permanent Generation.
If heap space is full, Java throws *java.lang.OutOfMemoryError*.
Access to this memory is comparatively slower than stack memory
This memory, in contrast to stack, isn't automatically deallocated. It needs Garbage Collector to free up unused objects so as to keep the efficiency of the memory usage.
Unlike stack, a heap isn't threadsafe and needs to be guarded by properly synchronizing the code.

JVM vs. JDK vs. JRE

- Please read and understand the below article:

https://www.baeldung.com/jvm-vs-jre-vs-jdk

Memory Models

- Code tracing and warm up:

```
int var1;
var1 = 52;
int var2;
var2 = var1
var1 = 127;
System.out.println("var1 is " + var1 +", var2 is " +
var2);
```

Memory Models

Primitive types vs. Object types

Primitive Types: boolean, byte, short, int, long, float, double, char
Object Types: Arrays and classes

```
int var1;
var1 = 123;
Location sjsu;
sjsu = new Location(37.33578624343321, -121.88006997482594);
Location goldenGate = new Location(37.80799311665714, -
122.47526935570986);
goldenGate.setLatitude(-123);
```

Memory Models

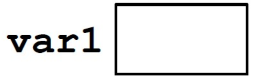
```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

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

Memory Models

```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

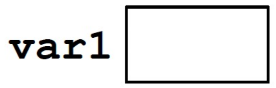
Variable declaration: draw a box and label it with the variable's name



Memory Models

```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

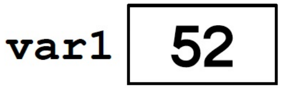
<u>Variable assignment</u>: put the value of the right hand side (RHS) into the box for the variable on the left hand side (LHS)



Memory Models

```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

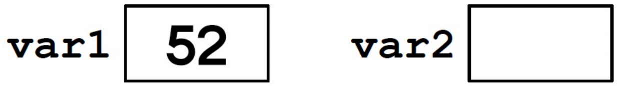
Variable assignment: put the value of the right hand side (RHS) into the box for the variable on the left hand side (LHS)



Memory Models

```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

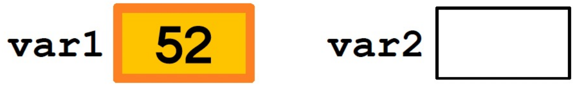
Variable declaration: draw a box and label it with the variable's name



Memory Models

```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

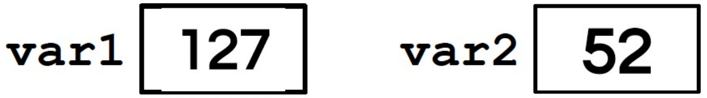
Variable assignment: put the value of the right hand side (RHS) into the box for the variable on the left hand side (LHS)



Memory Models

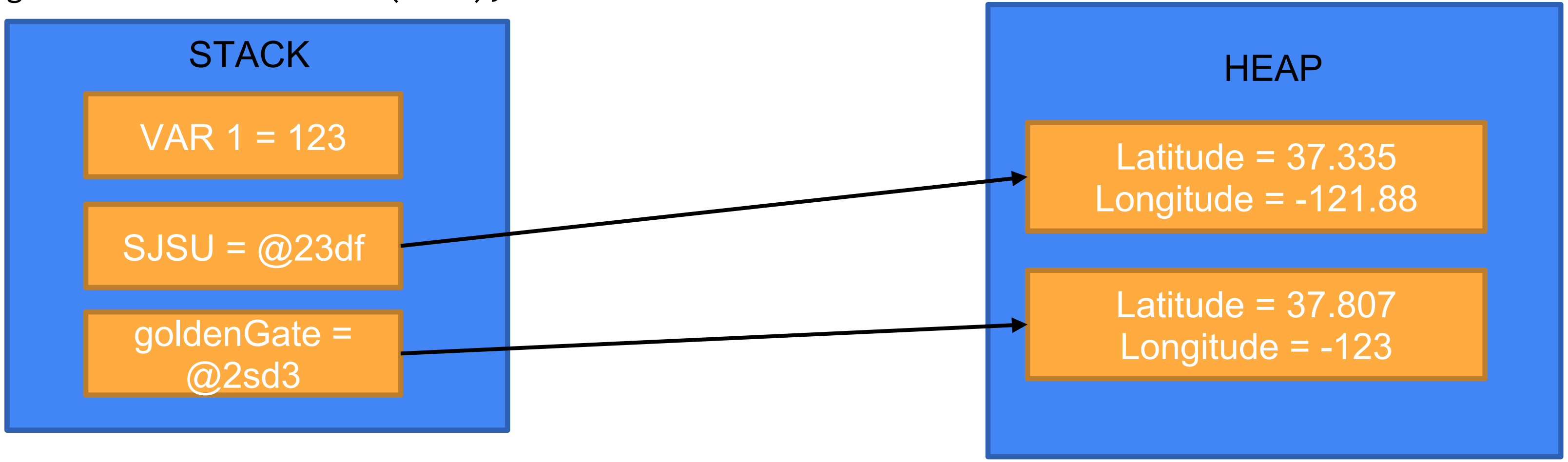
```
int var1;
var1 = 52;
int var2;
var2 = var1;
var1 = 127;
System.out.println("var1 is " + var1 +
                   ", var2 is " + var2);
```

<u>Variable assignment</u>: put the value of the right hand side (RHS) into the box for the variable on the left hand side (LHS)



Memory Model (First example)

```
int var1;
var1 = 123;
Location sjsu;
sjsu = new Location(37.33578624343321, -121.88006997482594);
Location goldenGate = new Location(37.80799311665714, -122.47526935570986);
goldenGate.setLatitude(-123);
```



REFERENCES

- Constructors: https://www.geeksforgeeks.org/constructors-in-java/
- https://www.baeldung.com/java-stack-heap
- https://www.programiz.com/java-programming/method-overloading
- https://www.baeldung.com/jvm-vs-jre-vs-jdk