Pillars of OOP

2.1 - Classes and Objects

Class

- Non primitive or user-defined data type in Java
- A blueprint or template for objects that share a set of attributes/properties/characters, a set of behavior/methods/actions
- It is a logical entity that does not occupy any space/memory - memory is allocated when we create an instance of the object

Constructor

- Constructors are special methods whose name is the same as the class name
- The constructors serve the special purpose of initializing the objects - no-arg constructor, no parameters
- Parameterized constructor = constructors that take some arguments
- Copy constructor = in Java we define copy constructor on our own
- Constructor chaining (this() - means the other constructor, super() - can be called in a child constructor

Example

```java
class A { //implicitly extends object
    int m_x;
    int m_y;
    int m_z;

    public A (int x, int y, int z) {
        super();
        m_x = x;
        m_y = y;
        m_z = z;
        this(); // recursive constructor invocation error
    }

    public int A() { // not a constructor, compiles but not       // reccomended
        return m_x;
    }

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

Java Destructor

- The destructor is the opposite of the constructor
    - The destructor is used to delete or destroy the object that releases the resource occupied by the object
- There is no concept of destructor in Java
    - 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
    - Garbage Collection - memory is finite, objects allocated to the heap must …
- …

Pillars of Object Oriented Programming

1 - Data Abstraction

Definition

The process of hiding certain (unnecessary) details and showing only essential information to the user

Achieving Abstraction

Abstract Classes

- Partial abstraction

```
abstract class A {
    abstract void foo()
}
```

Interfaces

- Full abstraction

```
public interface Player {
    public static final int League = 1;
    abstract int move();
}
```

- Classes (not abstract, object can be instantiated from the class)
- Abstract Methods

ⓘ **Marker Interface**

- Empty interface

⚠ **Warning**

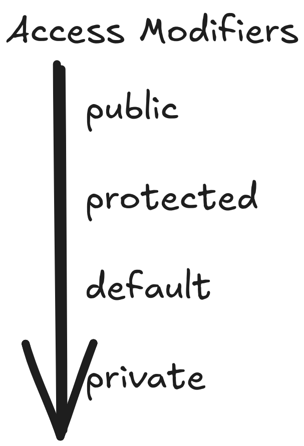
An instance of an abstract class cannot be created

2 - Encapsulation

ⓘ **Definition**

Binding data and functions that manipulate data in a single unit (class)

- Enables data-hiding, [abstraction](), [data access control]()



3 - Inheritance

ⓘ **Definition**

Allows code reuse and enables polymorphism

Example

```
Student s = (Student) new Person(); // will compile, but explicit type casting is not reccommended
s.getGpa(); // s is not a student so JVM will find an error
```

4 - Polymorphism

Runtime polymorphism

Method overriding

Method Overloading

Compile time polymorphism

Method overloading or operator overloading

- Multiple methods with the same name but different signatures
  - Different number or types of arguments
- We cannot overload by return type
  - void foo() and int foo()
- Static methods can be overloaded
- If I have the standard main() function with String[] args, jvm will detect that as an entry point
  - If I additionally make a main() function with different parameters or no parameters, it will just be a function
- Operator overloading
  - Java doesn't allow user-defined operators
  - Internally Java overloads some operator
  -
    - can be arithmetic addition or String concatenation
- Overload vs Override
  - Overloading
    - Compile time polymorphism
    - same class

- Overriding
    - Run time polymorphism
    - Superclass/subclass

Memory Regions

- Hard disk is too slow so the program is loaded to memory
- A program's memory usage typically includes 4 different regions
    - Code memory - program instructions are stored here
        - constants
    - Static memory
        - static fields
    - Heap/Stack
        - Order of adding data in stack and heap matters!!!
        - Stack (automatic memory)
            - local (instance) variables are allocated during a method call
            - a method call adds local variables to the stack, and a return removes them
            - like adding and removing dishes from a pile; hence the term "stack
            - automatically allocated and deallocated
            - contains references to heap
        - Heap (free store)
            - where the "new" operator allocates memory for objects

Hierarchy

- Every constructor of every class begins with super(). If we do not provide super, it will automatically make an implicit no-args ctor call to the superclass
- If you call super and there is not a no arg constructor, then it will be a compiler error, other ctors will still work