Under-Grad Course on

Java Programming

21-805-0205: Java Programming

- **Java Basics**: History of Java, Java features, data types, variables, operators, expressions, control statements, type conversion and casting, Concepts of - classes, objects, constructors, Access Specifiers, Access Modifiers, overloading methods, recursion, nested and inner classes
- **Inheritance**- Inheriting data members and methods, Single and Multilevel inheritance, use of super and this keywords. **Polymorphism**- method overriding, dynamic method dispatch, abstract and final classes. **Arrays and Strings** - One dimensional arrays, Multidimensional arrays, exploring String class and methods, String Buffer class. **Interface**: creation and implementation of an Interface. **Packages** - creating and accessing a package, importing packages, creating user defined packages
- **Exception Handling**: benefits of exception handling, exception hierarchy, usage of try, catch, throw, throws and finally, built-in exceptions, creating own exception sub classes. **Multi-threaded Programming**: thread life cycle, creating threads, thread priorities, synchronizing threads, Inter Thread Communication.
- Managing **input/output** files in java, concepts of streams, stream classes, byte stream classes, character stream classes, using streams, I/O classes, file classes, I/O exceptions, creation of files, reading/writing characters, byte handling primitives, data types, random access files, JDBC (Java Database Connectivity), overview, implementation.
- **Java generics**- boxing and unboxing, varargs, subtyping, wildcards, reifiable types, reflected types; **Java collections** - Collection framework, Collection interfaces, Sets - HashSet, TreeSet, Queues- PriorityQueue, BlockingQueue, Lists- ArrayList, Maps - HashMap, TreeMap, ConcurrentMap, **Java lambdas**- functional interfaces

- **References**

  - 1. C. Thomas Wu, An introduction to Object-oriented programming with Java, 5e, McGraw-Hill, 2009.
  - 2. Cay S. Horstmann, Core Java: Volume I – Fundamentals, 11e, Pearson, 2020.
  - 3. Herbert Schildt, Java: The Complete Reference, 9e, McGraw-Hill, 2017.
  - 4. K. Arnold, J. Gosling, David Holmes, The JAVA programming language, 4e, Addision-Wesley, 2005.
  - 5. Paul Deitel and Harvey Deitel, Java, How to Program: Early Objects, 11e, Pearson Education, 2018.
  - 6. Timothy Budd, Understanding Object-oriented programming with Java, 2e, Pearson Education, 2001.
  - 7. Y. Daniel Liang, Introduction to Java programming, Comprehensive Version, 10e, Prentice Hall India, 2013.
  - 8. Bruce Eckel, Thinking in Java, 4e, Prentice Hall, 2006.

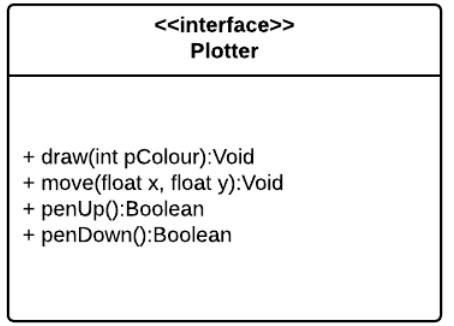
Objects

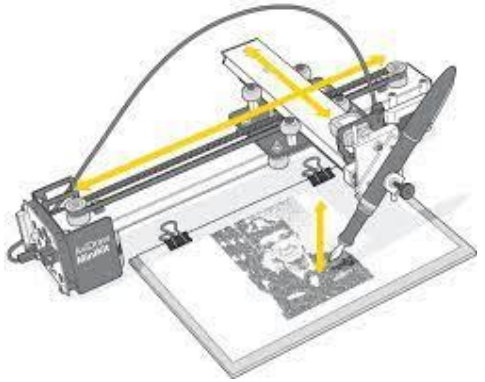
- Abstraction that resembles the problem than the solution (*computer*)
  - Everything is an object
  - Create composite objects by extending simple objects
  - Each object has its own memory (state)
  - A program is a collection of objects and messages sent between them
  - Objects are instances of classes (types)

Interface

- How does an object send a message (make a request) to another object?
  - What are the specific behaviours that the destination object is designed to perform?
  - Known to other objects

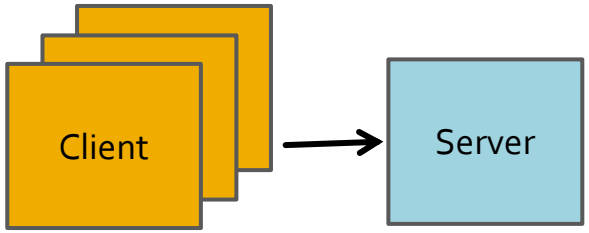
```
class CallingClass implements Plotter
{
    public static void main(String[] args)
    {
        Plotter pltr = new Plotter();
        if (pltr.penDown())
        {
            pltr.draw(2);
        }

    }
}
```





Philosophy of OOP

- To solve *your* problem, create (or even better, locate and include) objects that provide the required services.
  - How to decompose the problem to a set of objects?
    - Find whether the objects already exist and accessible. Reuse them.
    - Create only the remaining set of objects. KISS.
    - Build a cohesive solution.
    - Larger implementation settings: Hide the implementation while exposing the interfaces.



Access Specifiers and Packages

- Hiding at different levels of granularity
  - Public
  - Private
  - Protected
  - Default
- Modular design: Package access

Open/Closed Principle

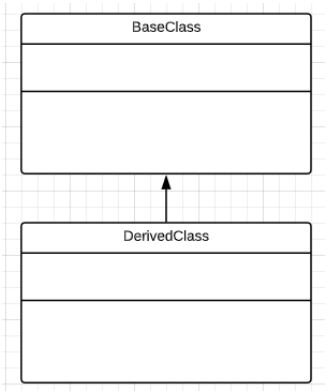
- Entities should be open for extension, but closed for modification

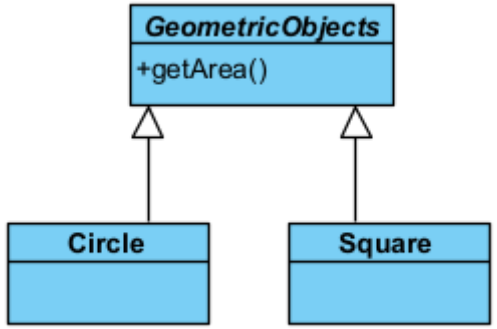
```
interface GeometricObjects {

   // method to be implemented
    public double getVolume();
}

class Cuboid implements GeometricObjects {

    // props of a cuboid

    public double length;
    public double breadth;
    public double height;

    // function implementation to calculate
    // the volume of a cuboid
    public double getVolume()
    {
        return length * breadth * height;
    }
}
```

```
class Sphere implements GeometricObjects {

    // props of a sphere
    public double radius;

    // function implementation to calculate
    //  the volume of a sphere
    public double getVolume()
    {
        return (4 / 3) * Math.PI * radius * radius * radius;
    }
}
```

Inheritance

- Inherit by extension
  - Base class
  - Derived class





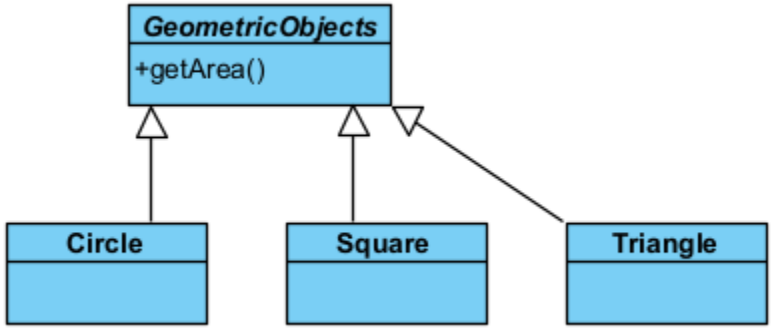

```
class abstract GeometricObjects {

    // abstract method to be implemented
    public abstract double getArea();
}

class Square extends GeometricObjects {

    // class property
    public double side;

    // function implementation to calculate
    // the area of a square
    public double getArea()
    {
        return side * side;
    }
}
```

```
class Circle extends GeometricObjects {

    // class property
    public double radius;

    // function implementation to calculate
    // the area of a circle
    public double getArea()
    {
        return Math.PI * radius * radius;
    }

    // function implementation to calculate
    // the perimeter of a circle
    public double getPerimeter()
    {
        return 2 * Math.PI * radius;
    }
}
```

Generic Base Class

- Dynamic binding

```
Triangle triangle = new Triangle();
getArea(triangle);
```




```
class Triangle extends GeometricObjects {

    // class properties
    public double base;
    public double altitude;

    // function implementation to calculate
    // the area of a triangle
    public double getArea()
    {
        return 0.5 * base * altitude;
    }
}
```

Object and (its) Reference

- To manipulate objects, take them with you

```
String str = "abcd"; //o|o
// You still carry your old C with you?

String str = new String("abcd"); //That is it! :)
str.append("efgh");
```

- Where do the objects live?
  - On the Heap memory
    - *java.lang.OutOfMemoryError*

Primitive Types

Architecture-agnostic

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	**Boolean**
char	16 bits	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8 bits	-128	+127	**Byte**
short	16 bits	-2^{15}	$+2^{15}-1$	**Short**
int	32 bits	-2^{31}	$+2^{31}-1$	**Integer**
long	64 bits	-2^{63}	$+2^{63}-1$	**Long**
float	32 bits	IEEE754	IEEE754	**Float**
double	64 bits	IEEE754	IEEE754	**Double**
void	—	—	—	**Void**

Ultimate Base Class: Object

- Single root
- Scope of objects
- *Static*

Hello World

- File name?

```
// import libraries here

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

- Compile (Build)
- Execute (Run)
- Code Conventions
  - https://www.oracle.com/java/technologies/javase/codeconventions-filenames.html

Flow Control

- foreach iteration

```
public class Something {
  public static void main(String[] args) {
    int f[] = new int[5];
    for(int i = 0; i < 5; i++)
      f[i] = i;
    for(int x : f)
      System.out.println(x);
  }
}
```

- More....

Method Overloading

- All methods are ‘named’
- Constructor overloading
  - Chaining rule
- Methods overloading

```
void f() { ... }
void f(int x) { ... }
float f(float x) { ... }
float f(int x, float y) { ... }
```

- Overriding
  - Both the superclass and the subclass must have the same method name, the same return type and the same parameter list.
  - super keyword
  - access specifier restriction in overloading

Type Promotion Scheme

- double ← {int, long, float}
- int ← {char, short}
- short ←byte
- long ← int
- float ← int

Quiz

```
class OverloadingDemo {
  void add(int a,int b) {System.out.println("method#1 invoked");}
  void add(long a,long b) {System.out.println("method#2 invoked");}

  public static void main(String args[]) {
  OverloadingDemo od = new OverloadingDemo();

  od.add(20,20000000L);
  /* compiler sees an int first argument and
     a long second argument. Confused? */
  }
}
```

Object reference using this keyword

- To explicitly use the reference to the *current object*
  - To pass itself (object) to a foreign method, current method must use this
  - Nesting constructors

```
public class Flower {
  int petalCount = 0;
  String s = "initial value";
  Flower(int petals) {
    petalCount = petals;
    print("Constructor w/ int arg only, petalCount= "+ petalCount);
  }
  Flower(String ss) {
    print("Constructor w/ String arg only, s = " + ss);
    s = ss;
  }
  Flower(String s, int petals) {
    this(petals);
    this.s = s;
    print("String & int args");
  }
  Flower() {
    this("hi", 47);
    print("default constructor (no args)");
  }
  void printPetalCount() {
    print("petalCount = " + petalCount + " s = "+ s);
  }
  public static void main(String[] args) {
    Flower x = new Flower();
    x.printPetalCount();
  }
}
```

this inside Constructor

- Difference between object reference and constructor reference

```
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 1, 1);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    // rest of the code here
}
```

Initialization

```
boolean state = true;
char ch = ‘a’;
byte b = 47;
short s = 0x0f;
int i = 10;
long lng = 12345678910;
float f = 3.14f;
double d = 3.141592653589793;
```

boolean	false (0)
char	[ ]
byte, short, int, long	0
float, double	0.0
object	null

- Order of initialization
  - Within a class, variables are initialized before any methods (including the constructor) can be called
  - Instance initialization

Instance Initialization

```
class Cup {
  Cup(int marker) {
  System.out.println("Cup(" + marker + ")");
  }
}

public class Cups {
  Cup Cup1;
  Cup Cup2;
  {
    Cup1 = new Cup(1);
    Cup2 = new Cup(2);
    System.out.System.out.println("Cup1 & Cup2 initialized");
  }
  Cups() {
    System.out.System.out.println("Cups()");
  }
  Cups(int i) {
    System.out.System.out.println("Cups(int)");
  }
  public static void main(String[] args) {
    System.out.System.out.println("Inside main()");
    new Cups();
    System.out.System.out.println("new Cups() completed");
    new Cups(1);
    System.out.System.out.println("new Cups(1) completed");
  }
}
```

Array Initialization

```
int[] array;

int array [];

int[] array = { 1, 2, 3, 4, 5 };

for(int i = 0; i < array.length; i++)
  System.out.printlnln(array[i]);
```

- Autoboxing and Unboxing
-

```
Integer[] a = {
  new Integer(1), new Integer(2), 3
};
```

Autoboxing

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
    li.add(i);
    /* you don't need to convert as
    li.add(Integer.valueOf(i)); */
```

Implementation Hiding

- Separation of concerns
  - Client-side programmers v/s Server-side programmers
  - Libraries
  - Packages
    - The first non-comment line

```
package com.example.mypackage;

public class MyClass {
  // code here
}


import com.example.mypackage.MyClass; // use this library in the code
```

      - Where is this java file located?
      - Deployment of large libraries