Lecture 7

Static/Non Static

Reminders

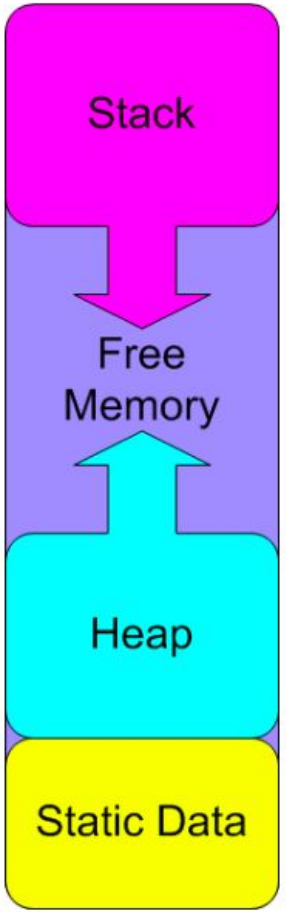
- Mic

Memory

- **Stack** - The stack is for local variables and for maintaining a record of function calls. The stack grows from the top of memory down towards the heap.

- **Heap** - The heap is for dynamically allocated data items. The heap grows from the top of the static data area up as data items are allocated.

- **Static Data** - This is a block of reserved space in RAM for all the global and **static** variables from your program. Allocated once and lasts for duration of a program.



Static Method, example

```
public class Chalk {
     public String color;
     public Chalk(String cr) {
          color = cr;
     }

     public static void write(String word) {
          System.out.println(word);
     }
}
```

```
public class ChalkTest {

  public static void main(...) {

     Chalk.write("hi");

 }

}
```

Output?

A: “hi”
B: Compile Error

Static Method, example 2

Error: non-static variable color cannot be referenced from a static context

```
public class Chalk {
     public String color;
     public Chalk(String cr) {
          color = cr;
     }

     public static void write(String word) {
          System.out.println(word);
     }
}
     public static void printColor() {
          System.out.println(color);   <-non-static.
}
```

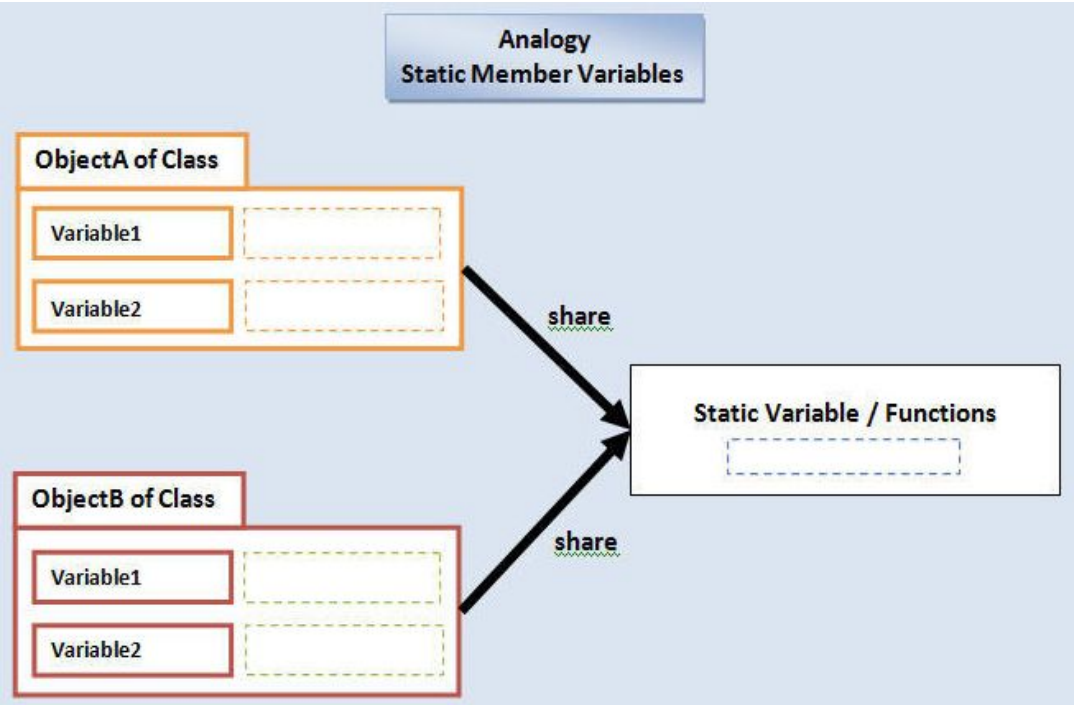
```
main(String []args) {

     Chalk.printColor();
}
```

Static: Belongs to the class

- Class creates objects
- Constructor makes them unique (if needed)
- How to create a property that all objects share?
  - You do not want to modify this property for each object. You may want to change it just once!
  - Example: number of the cars sold, sound of the horn

Shared properties and methods



Chalk again. 1

```
public class Chalk {
      public String color;
      public Chalk(String cr) {
           color = cr;
     }
      public void printColor() {
           System.out.println(color);
     }
}
```

```
Chalk ch1 = new Chalk("white");
Chalk ch2 = new Chalk("black");

ch2.printColor();

ch1.printColor();
```

A: white
   white

B: black
   black

C: black
   white

D: white
   black

E: Something else

Chalk again. 2

```
public class Chalk {

      public static String color;

      public Chalk(String cr) {

            color = cr;

      }

      public void printColor() {

            System.out.println(color);

      }


}
```

```
Chalk ch1 = new Chalk("white");
Chalk ch2 = new Chalk("black");

ch2.printColor();

ch1.printColor();
```

A: white
   white

B: black
   black

C: black
   white

D: white
   black

E: Something else

static variables: Common to all instances (One fixed memory location)

```
public class HelloWorld{
     int count;
     public HelloWorld(int i) {
          count = i;
     }

 public static void print(){
     System.out.print(count);
 }
}
```

```
HelloWorld test = new HelloWorld(1);
HelloWorld test2 = new HelloWorld(2);
test.print();
```

What will be printed?

A: 1

B: 2

C: Can’t predict

D: void

E: Error

What gets printed

```
public class Car
{
  public static int numCars = 0;
  public String color;
  public Car(String cl) {
    numCars++;
    color = cl;
 }
 public static int getNumCars() {
   return numCars;
 }
}
```

```
public class Test {
     public static void main(…) {
      Car c1 = new Car(“white”);
      Car c2 = new Car(“green”);
      Car c3 = c1;
      int i = c3.getNumCars();
      System.out.println(i);
     Car c4 = new Car(“blue”);
     }
}
```

A) Compiler error
B) 0
C) 1
D) 2
E) 3

What gets printed

```
public class Animal {
  public static String color;
  public Animal (String c) {
    color = c;
  }
  public String getColor() {
   return color;
}
```

```
public class Driver {
   public static void main(…) {
     Animal a1 = new Animal(“blue”);
     Animal a2 = new Animal(“purple”);
     System.out.println(a1.getColor());
     System.out.println(a2.getColor());
   }
}
```

A: blue, blue
B: blue, purple
C: purple, blue
D: purple, purple
E: Error

Expected output?

```
class Example {
    public void test() {
        int a = 0;
        int b = 1;
        System.out.print(a + b);
    }
    public static void main(String[] args) {
        test();
    }
```

A: 0

B: 01

C: 1

D: Error

public static void main(String[] args)

Mystery of

```
public static void main (String  [] args)
```

is revealed

```
public static void main (String  [] args)
```

```
public static void main (String  [] args)
```

```
public static void main (String  [] args)
```

```
public static void main (String  [] args)
```

One Special Role for Strings: Command Line Arguments

```
public class ArgsDemo {
    /** Prints out the 0th command line argument. */
    public static void main(String[] args) {
        System.out.println(args[0]);
    }
}
```

```
>>> java ArgsDemo hello some args
hello
```

ArgsSum Exercise

Goal: Create a program ArgsSum that prints out the sum of the command line arguments, assuming they are numbers.

ArgsSum Exercise

Goal: Create a program ArgsSum that prints out the sum of the command line arguments, assuming they are numbers.

```
public static void main(String args[]) {

    int sum = 0;
    for (int i = 0; i<args.length; i++){
        sum = sum + Integer.parseInt(args[i]);
    }
    System.out.println(sum);
}
```

Question + Demo

Every variable in Java must be declared and initialized in order to have a meaningful value.

A: Of course! (True)

B: Not at all! (False)

THIS
SETTERS AND GETTERS

this

- *this* is a keyword in Java.
- Can be used inside method or constructor of a class.
- It(**this**) works as a *reference* to a current object whose method or constructor is being invoked.
- Similar to **self** in Python.

```
public class Tester {
    int a;
    int b;

    Tester(int a, int b) {
       this.a = a;
       this.b = b;

    }
    void display() {
        System.out.println("a = " + a + "  b = " + b);
    }

    public static void main(String[] args) {
        Tester object = new Tester(10, 20);
        object.display();
    }
}
```

https://www.geeksforgeeks.org/this-reference-in-java/

Setters and Getters

```
class MyAge{
    public String name;
    public int age;

    public MyAge(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public static void main(String args[]) {
        MyAge me = new MyAge("Marina", 20);
        System.out.println(me.age);
    }
}
```

Setters and Getters

Output: 20

```
class MyAge{
    public String name;
    public int age;

    public MyAge(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public static void main(String args[]) {
        MyAge me = new MyAge("Marina", 20);
        System.out.println(me.age);
    }
}
```

Setters and Getters

Output: 60

```
class MyAge{
    public String name;
    public int age;

    public MyAge(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public static void main(String args[]) {
        MyAge me = new MyAge("Marina", 20);
       Me.age = 60;
        System.out.println(me.age);
    }
}
```

Setters and Getters

Output: -100

```
class MyAge{
    public String name;
    public int age;

    public MyAge(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
    public static void main(String args[]) {
        MyAge me = new MyAge("Marina", -100);
        System.out.println(me.age);
    }
}
```

Setters and Getters

```
class MyAge{
    public String name;
    public int age;

    public MyAge(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
    public static void main(String args[]) {
        MyAge me = new MyAge("Marina", -100);
        System.out.println(me.age);
    }
}
```

Encapsulation

In short: hiding information from a user, only you have control over your code.