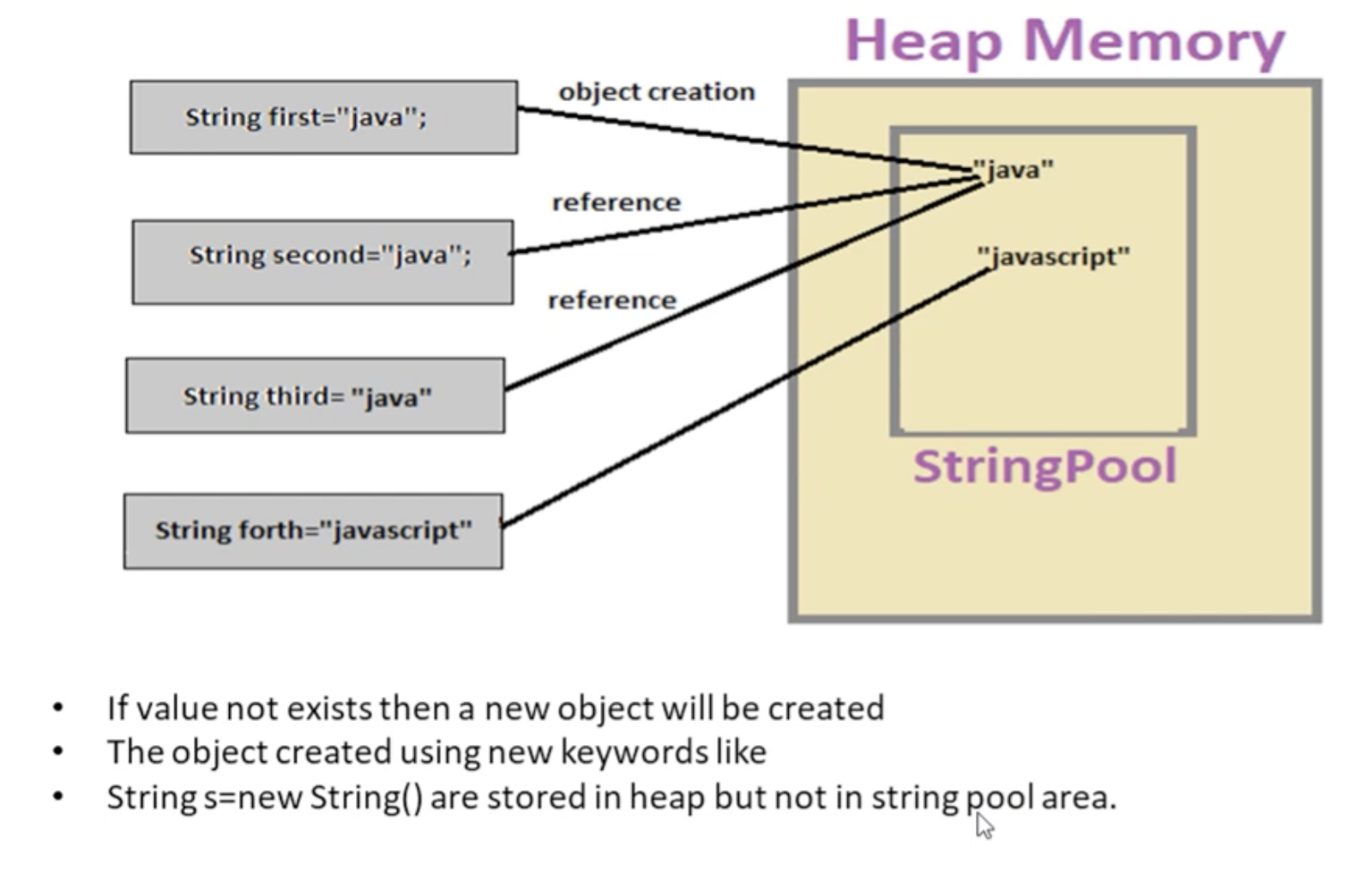
## 01 Java DataTypes and OOP Basics

* Primitive data types
  + Byte, short, int, long
  + Float, double
  + Char
  + Boolean
    - Circularity of values
    - Range of values and type mismatch error
    - Auto promotion
* Wrapper data types
  + Byte, Short, Integer, Long, Float, Double, Character, Boolean
* Immutability, pass by reference/value in Java
  + Code:
    - Trying to modify the value of immutable integer object code
* Strings, String Pools
  + 

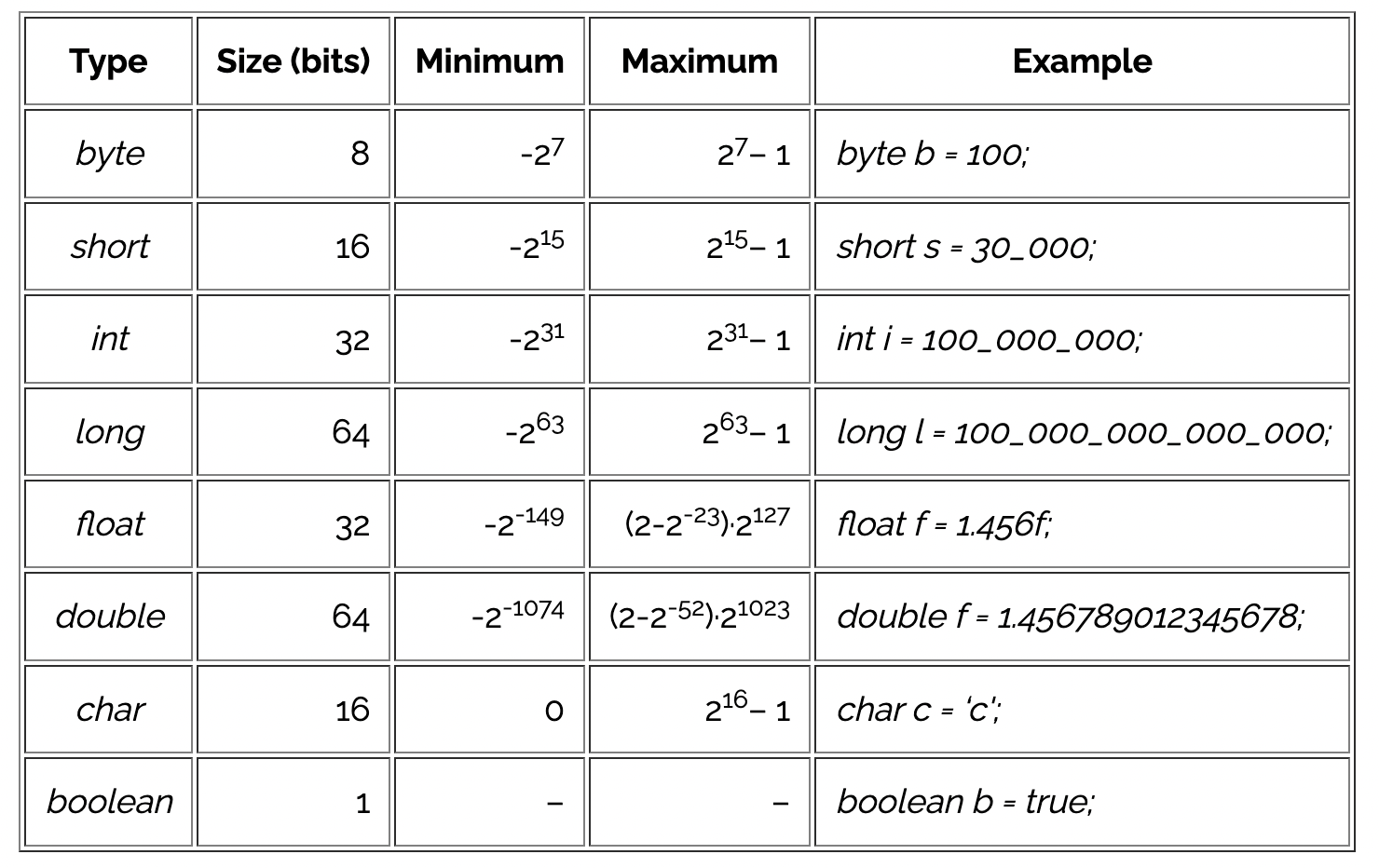
### Data types and Ranges

Primitive Data Types

The eight primitives defined in Java are int, byte, short, long, float, double, boolean and char. These aren't considered objects and represent raw values.

They're stored directly on the stack

We'll take a look at storage size, default values and examples of how to use each type



* + - Circularity of values  
      by=127  
      By+1 -> -128
    - Range of values and type mismatch error [demotion doesn’t happen automatically]  
      byte by = sh; // type mismatch error  
      byte by = in; // type mismatch error
    - Auto promotion  
      sh = by;

### Arrays and Strings

### Introduction

**Object Oriented Programming** (OOP) is a way of programming that helps us to write maintainable code. Maintainable code is:

1. Easy to understand
2. Easy to change
3. Not susceptible to errors.

OOP does so by mimicking real life interactions in code.

3 features of OOP:

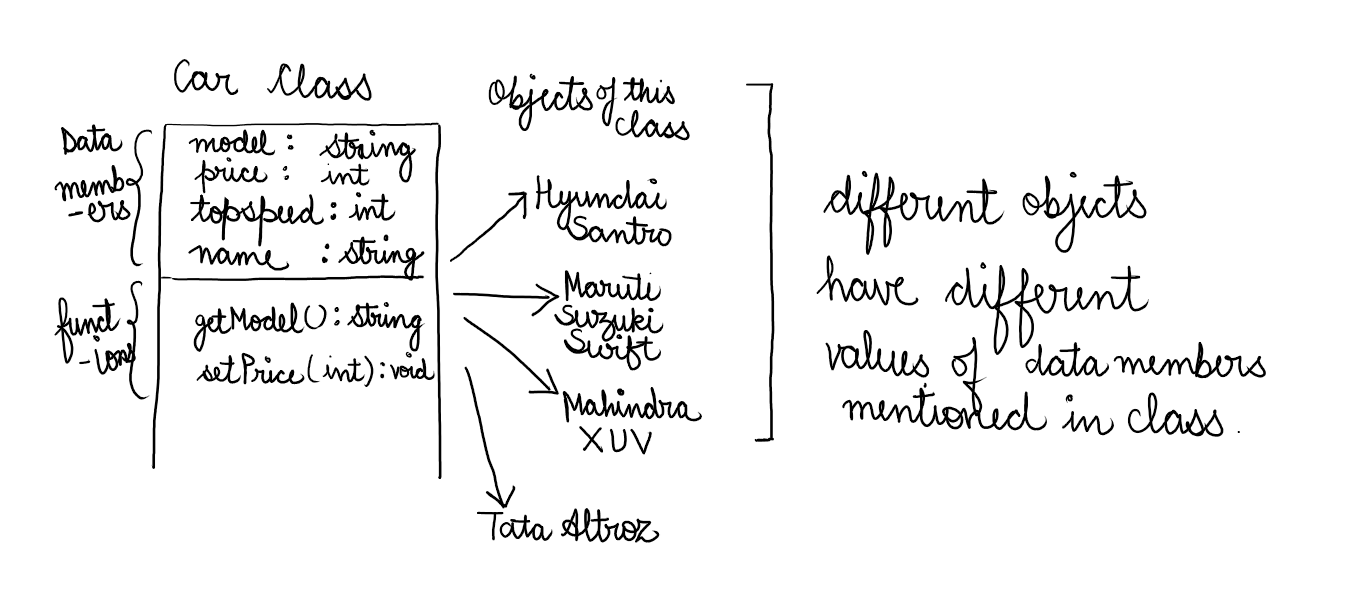
1. Encapsulation: Keeping information and functions together  
     
   Data members and member functions for doctor and mechanic objects are together combined in one entity or container
2. Abstraction: Hiding unnecessary details  
     
   Doctor doesn’t have to worry about how the mechanic repairs the car.
3. Message Passing: Communication between objects  
     
   Doctor objects calls function of mechanic object to get the car repaired.

### Classes and Objects

Now, let us understand about classes. From the above discussion, we can conclude that there can be many doctor as well as mechanic objects. And in real life also, we can observe so many doctors such as orthopaedics, surgeons and many others. Now, there is a lot of commonality between doctor objects. The commonality is called the **class**. All the objects originate from the class.

Class is the blueprint and the objects are instances of the blueprint.

Let us take an example of Car class and its objects:



Hence, class is the blueprint which consists of data members and functions. Objects are instances of class. Each object has a different value of data members mentioned in class.

### How to create Classes and Objects, how to access its variables/ data members and how to change values of data members

Class Code:

public class Student {

// information

// data members

private String name;

private int rollno;

// what it can do with this information

// functions/ methods

}

Driver Class to instantiate objects of this class:

public class StudentUse {

public static void main(String[] args) {

// TODO Auto-generated method stub

Student s1 = new Student();

System.out.println(s1.rollno + " " + s1.name);

/\*

The output will be 0 null, as these are the default values of int and string in Java

\*/

s1.rollno=1;

s1.name="Akshima";

System.out.println(s1.rollno + " " + s1.name);

/\*

The output will be 1 Akshima, as these are the default values of int and string in Java

\*/

}

}

### Access Modifiers

We are going to talk about private, public and default. They are used to restrict access to variables/ data members and member functions of a class.

Code of Student Class using access modifiers:

/\*

This code is used to restrict illegal setting of roll numbers. This ensures that roll number is never set to negative.

\*/

public class Student {

// information

// data members

private String name;

private int rollno;

// what it can do with this information

// functions/ methods

public void setrollno(int rollno) {

if (rollno < 0) {

return;

}

this.rollno = rollno;

}

public void setname(String name) {

this.name = name;

}

public int getrollno() {

return this.rollno;

}

public String getname() {

return this.name;

}

}

Driver Class:

public class StudentUse {

public static void main(String[] args) {

// TODO Auto-generated method stub

Student s1 = new Student();

System.out.println(s1.getrollno() + " " + s1.getname());

s1.setrollno(1);

s1.setname("Akshima");

System.out.println(s1.getrollno() + " " + s1.getname());

}

}

**Thing to remember**:

Private access modifier restricts access to the same class. Variables and functions with private access modifier can be accessed within the same class.

Default access modifier restricts access to the same package. Variables and functions with default access modifier can be accessed within the same package.

Public access modifier can be accessed from anywhere. Variables and functions with public access modifier can be accessed from anywhere.

### Constructors

If you closely look at following lines of code:

Scanner scn = new **Scanner(System.in)**;

Student st1 = new **Student()**;

The bold lines look like functions. If is actually a function. It is a special type of function called Constructor.

The job of the constructor is to allocate memory to the object and create it in the heap.

The default constructor creates object and gives default values to the data members

public Student () {

}

The access specifier can be either public or private or default. The constructor doesn’t have any return type and its name is the same as that of the class name.

Parameterised Constructor

Apart from the default constructor, we can give custom values as arguments and those arguments can be used to initialise the values of data members.

Note: 1. Java gives you a default constructor, even if you don’t write it.

2. If you write your own parameterised constructor, then Java deletes its default constructor.

3. You can have multiple custom constructors.

Interfaces and Abstract Classes: <https://www.baeldung.com/java-abstract-class>

<https://www.baeldung.com/java-interfaces>

Wrapper Classes  
Autoboxing

Each primitive data type also has a full Java class implementation that can wrap it. For instance, the Integer class can wrap an int. There is sometimes a need to convert from the primitive type to its object wrapper (e.g., using them with generics).

Luckily, Java can perform this conversion for us automatically, a process called Autoboxing:

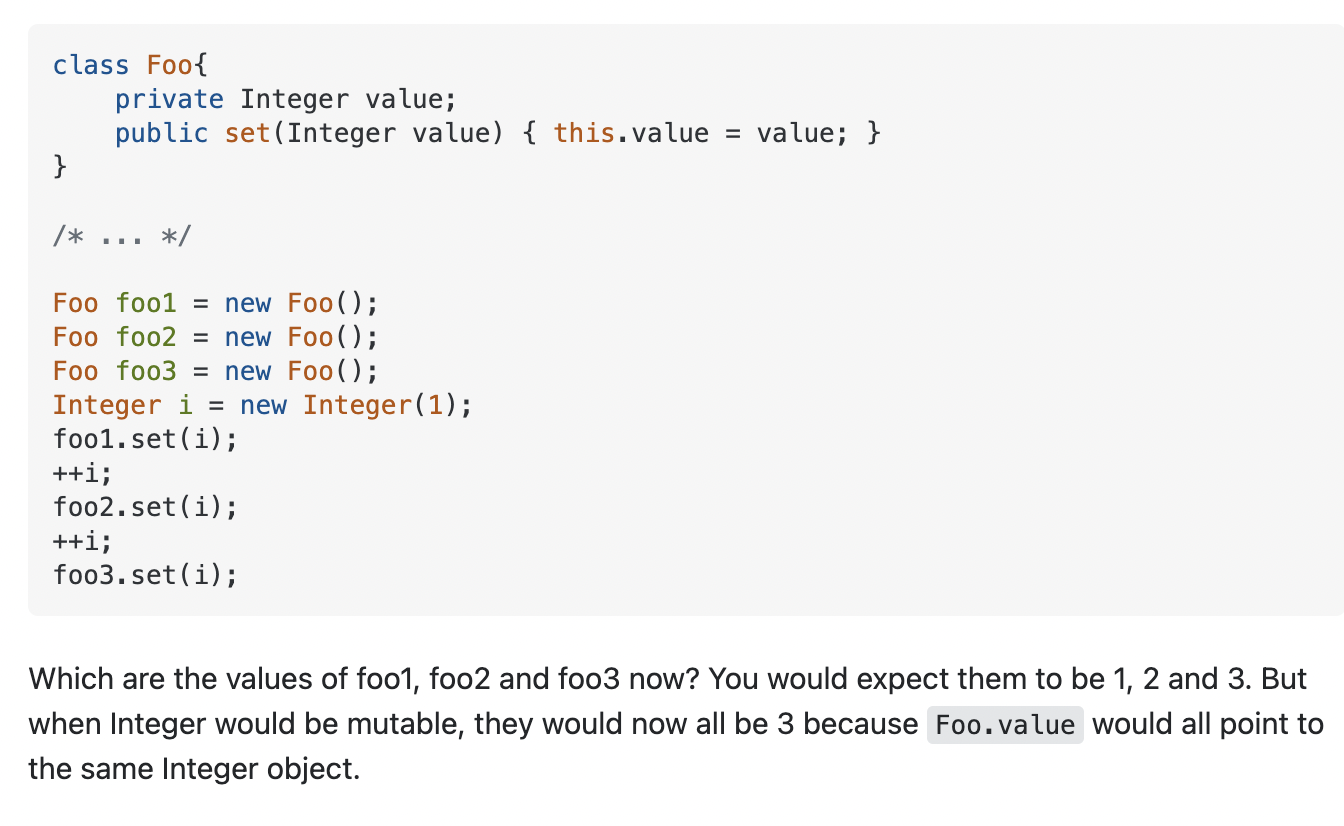
Character c = 'c';

Integer i = 1;

<https://stackoverflow.com/questions/36660904/java-wrapper-primitive-memory-allocation>

<https://stackoverflow.com/questions/12370544/why-are-java-wrapper-classes-immutable>

Memory map of immutability   
Immutability of Wrapper classes:



Pass by Reference and Value in Java: <https://www.scaler.com/topics/pass-by-value-and-call-by-reference-in-java/>