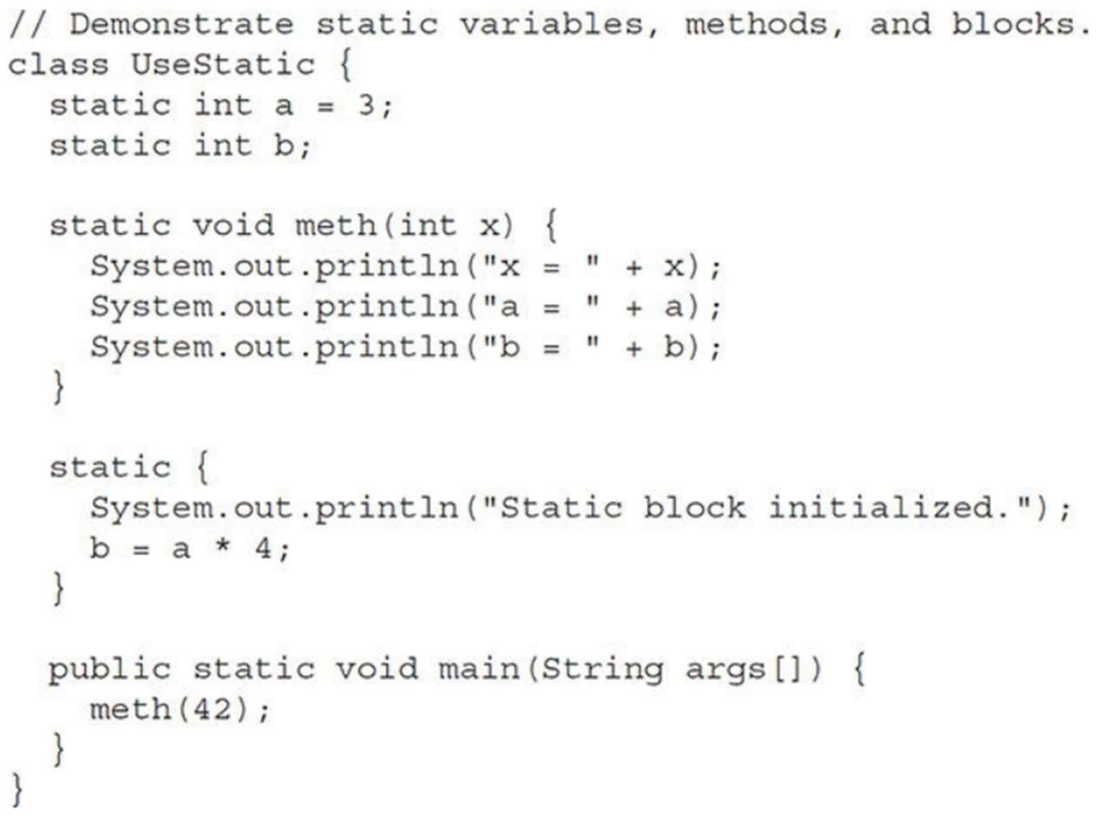
Understanding *static* member variables

- static member variable of class
    - Associated with class
    - Independent of object
    - One copy for all objects
    - Accessed without any object reference (Main Method)

Understanding *static* member variables

- Instance variables which are static: Global Variables:
  - One copy is shared among all objects
- Change the value of static variable of one object:
  - Change will be visible to all objects, because there is only one copy for all of them

Demo of static block



```
// Demonstrate static variables, methods, and blocks.
class UseStatic {
  static int a = 3;
  static int b;

  static void meth(int x) {
    System.out.println("x = " + x);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }

  static {
    System.out.println("Static block initialized.");
    b = a * 4;
  }

  public static void main(String args[]) {
    meth(42);
  }
}
```

Understanding *static* member methods

- Can only call other static methods:
  - Let’s Try to call Non-Static method
- Can access only static member variables of class:
  - Let’s Try to access non-static members
- Can be called on class and the object reference:
  - Let’s try both
- Can’t refer *this* and *super* keywords
- Example

How to access them

- Outside the class in which they are defined:
    - Classname.methodname();
    - Classname.datamember;

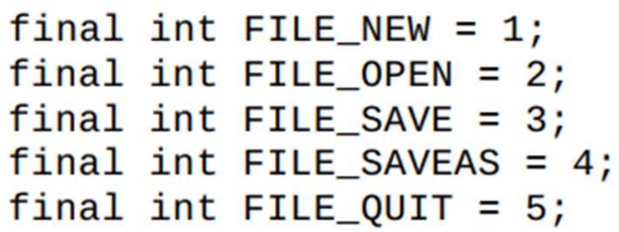
Understanding *static* code block

- Gets executed exactly once at the start

- Example (static member variable, method and block)

Introducing keyword *final*

- Usage with variable:
  - To make them constant
  - Variables must be initialized if made final
    - Initialized when declared/Initialized with the help of constructor



```
final int FILE_NEW = 1;
final int FILE_OPEN = 2;
final int FILE_SAVE = 3;
final int FILE_SAVEAS = 4;
final int FILE_QUIT = 5;
```

Introducing keyword *final*

- Usage with Methods:
  - Method parameters can be made final
  - Prevents them from being changed inside method body
  - Local variables can also be final
  - Final can be used with methods to prevent overriding, which we will discuss latter

Nested and Inner Classes

- Nested class
  - Class within a class
  - Two types of nested class, *static* and *non-static*
  - Scope of nested is bound with outer class
  - If B is inside A, B will not exist until A is created (Dependency)
  - *non-static* are common
- Example

Nested and Inner Classes

- Static class:
  - Declared inside a class
  - Using static keyword
  - Can have static/non-static data members
  - Can have static/non-static methods
  - Can not access the non-static members of outer class directly, will have to use an object
    - Because of it they are rarely used

Demo

- Create a nested static class
- Add some static/non-static members and methods
- Try to access the outer class variables without creating object of outer class
- Try to create object in Main Method

Nested and Inner Classes

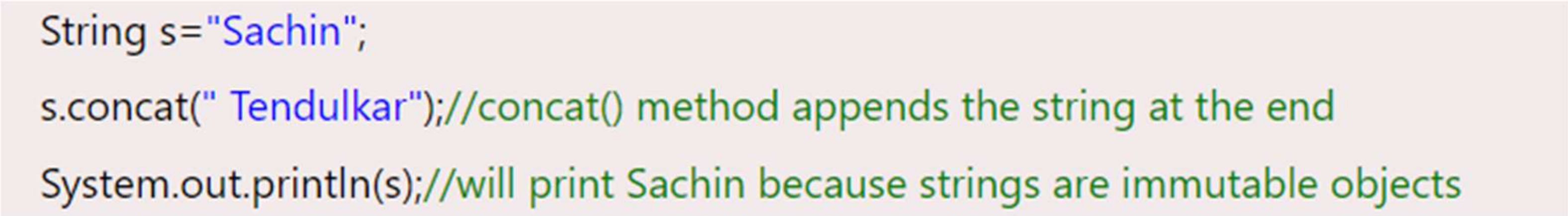
- Inner classes/ Non Static:
  - Nested/inner has access to member variables/methods including private of outer
  - Outer has no direct access of inner class methods/variables, an object of inner needs to be created

String class

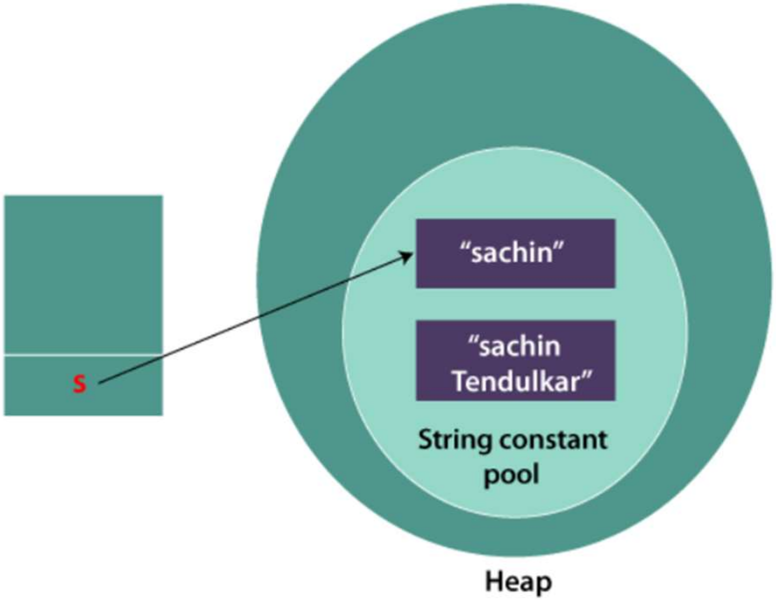
- Class in java.lang
- Even string constants inside System.out.print() are string objects
- String objects are immutable
- StringBuffer and StringBuilder are mutatable strings

Strings are Immutable!

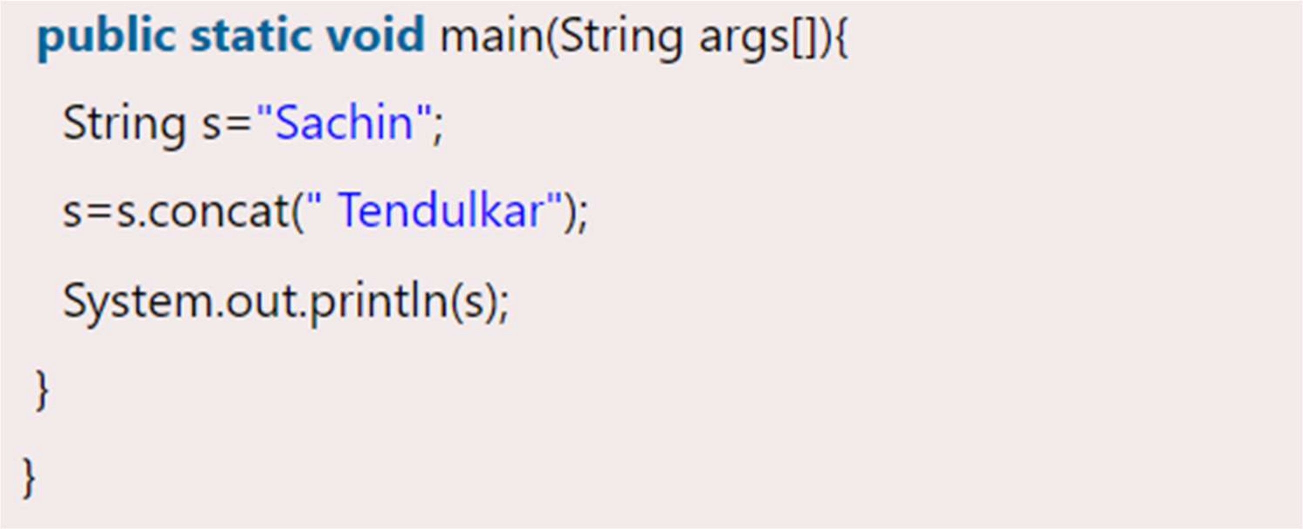
- Once you create a string, you can not change it’s value
- If you try to change the value a new string object is created
- But Why?
  - Strings are objects



```
String s="Sachin";
s.concat(" Tendulkar");//concat() method appends the string at the end
System.out.println(s);//will print Sachin because strings are immutable objects
```



Solution: Assign Reference to new string Object explicitly

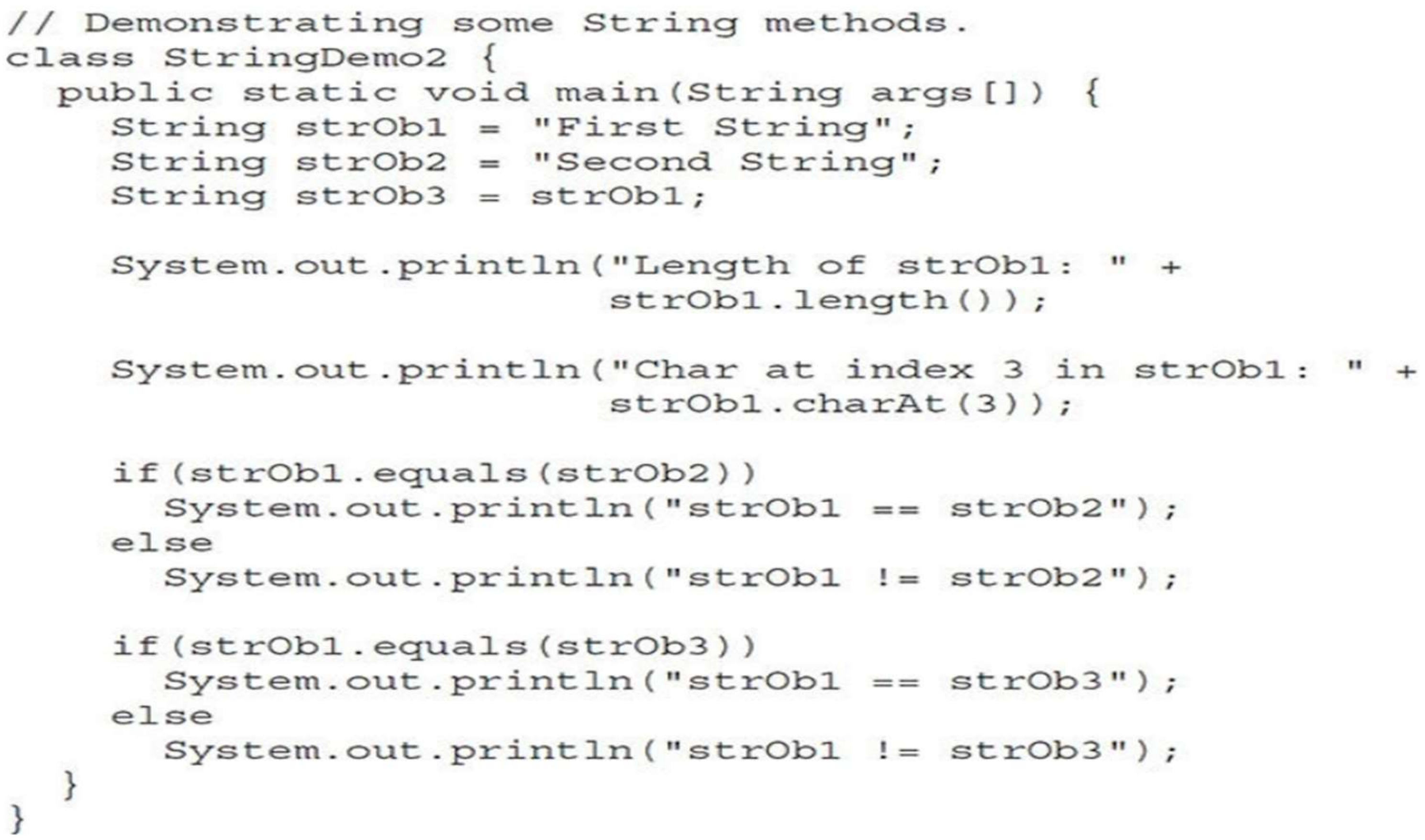


```
public static void main(String args[]){
  String s="Sachin";
  s=s.concat(" Tendulkar");
  System.out.println(s);
 }
}
```

String class

- String methods
  - equals()
  - equalsIgnoreCase()
  - length()
  - charAt()

Demo



```
// Demonstrating some String methods.
class StringDemo2 {
  public static void main(String args[]) {
    String strOb1 = "First String";
    String strOb2 = "Second String";
    String strOb3 = strOb1;

    System.out.println("Length of strOb1: " +
                       strOb1.length());

    System.out.println("Char at index 3 in strOb1: " +
                       strOb1.charAt(3));

    if(strOb1.equals(strOb2))
      System.out.println("strOb1 == strOb2");
    else
      System.out.println("strOb1 != strOb2");

    if(strOb1.equals(strOb3))
      System.out.println("strOb1 == strOb3");
    else
      System.out.println("strOb1 != strOb3");
  }
}
```

String class

- equals() method vs == operator
  - equals() checks value
  - == checks reference

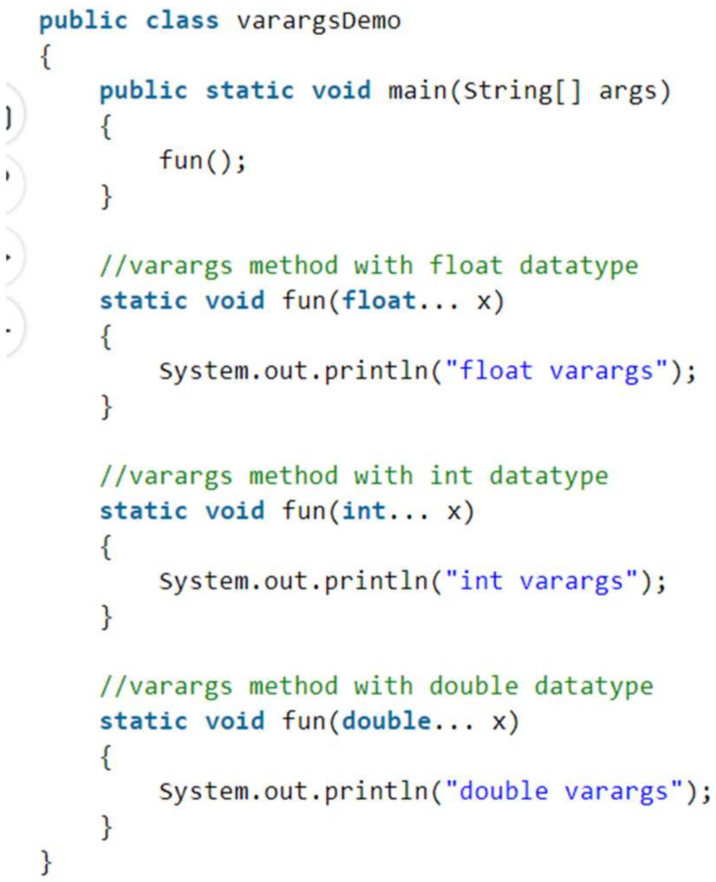
Varargs: Variable Length Arguments

- Support started with JDK 5
- If you are unsure about how many arguments a user will pass to method
- void add(int …arr):
  - Accepts zero or more arguments and places them in an array

Varargs: Variable Length Arguments

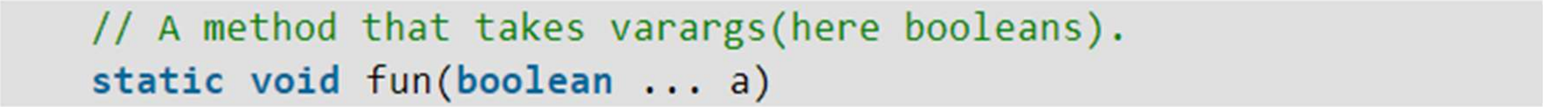
- We can have some mandatory arguments as well, so we need to place them before varargs
- Varargs is the last parameter of method

Varargs and Overloading

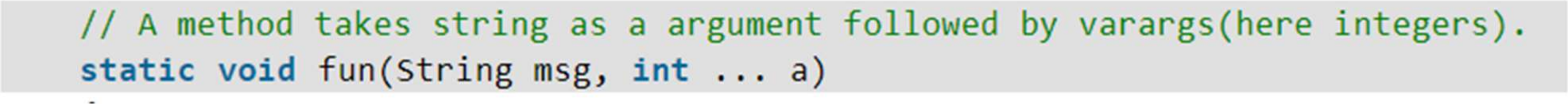


```
public class varargsDemo
{
    public static void main(String[] args)
    {
        fun();
    }

    //varargs method with float datatype
    static void fun(float... x)
    {
        System.out.println("float varargs");
    }

    //varargs method with int datatype
    static void fun(int... x)
    {
        System.out.println("int varargs");
    }

    //varargs method with double datatype
    static void fun(double... x)
    {
        System.out.println("double varargs");
    }
}
```

Varargs and Method Overloading

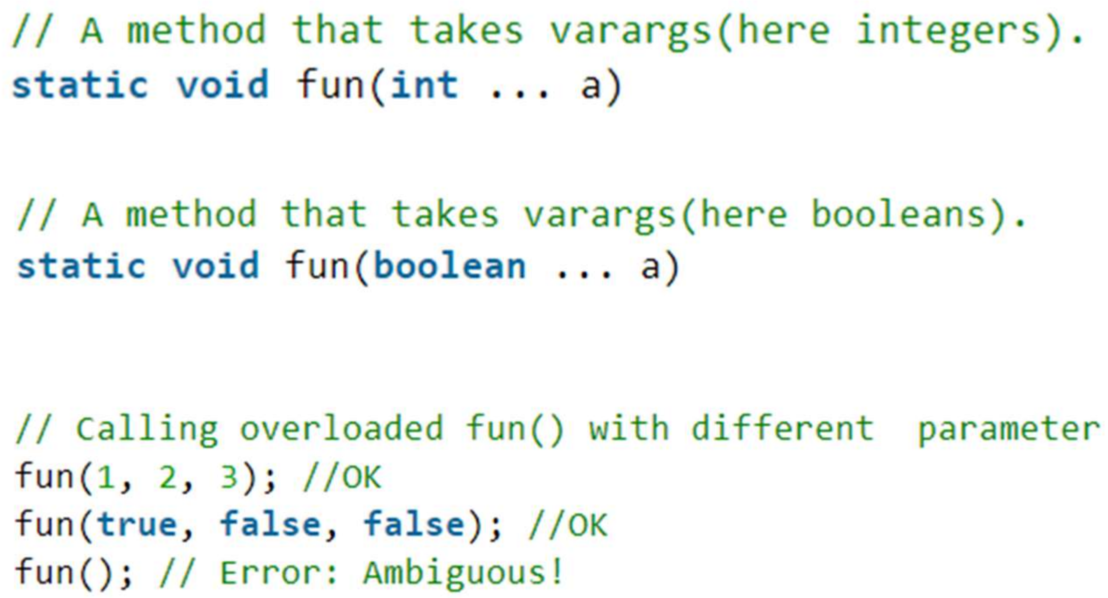


```
    // A method that takes varargs(here booleans).
    static void fun(boolean ... a)
```



```
    // A method takes string as a argument followed by varargs(here integers).
    static void fun(String msg, int ... a)
```

Varargs and Ambiguity

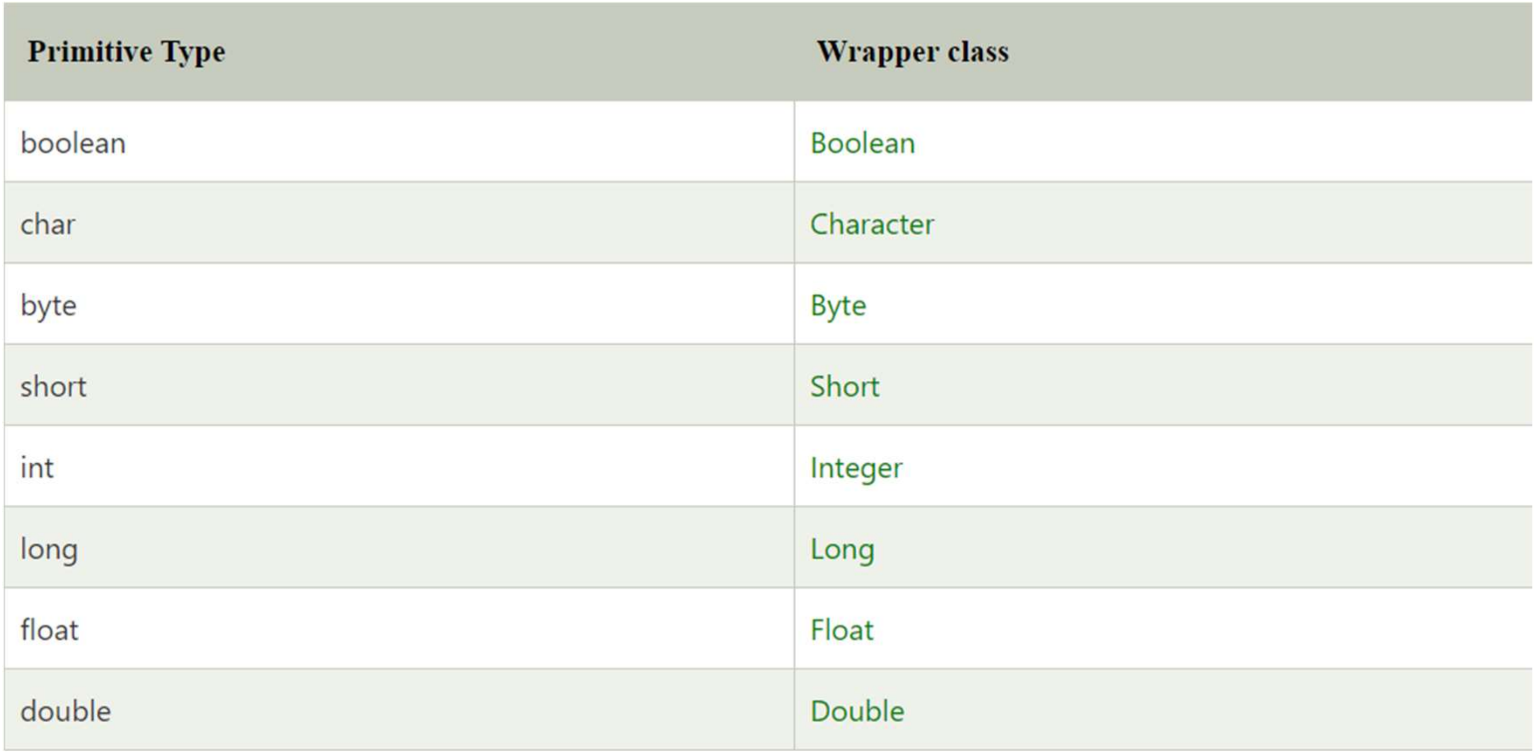


```
// A method that takes varargs(here integers).
static void fun(int ... a)

// A method that takes varargs(here booleans).
static void fun(boolean ... a)


// Calling overloaded fun() with different  parameter
fun(1, 2, 3); //OK
fun(true, false, false); //OK
fun(); // Error: Ambiguous!
```

Wrapper Classes

- provides the mechanism *to convert primitive into object and object into primitive*.
- AutoBoxing:
  - convert primitives into objects, automatically
- UnBoxing:
  - objects into primitives automatically

Wrapper Classes

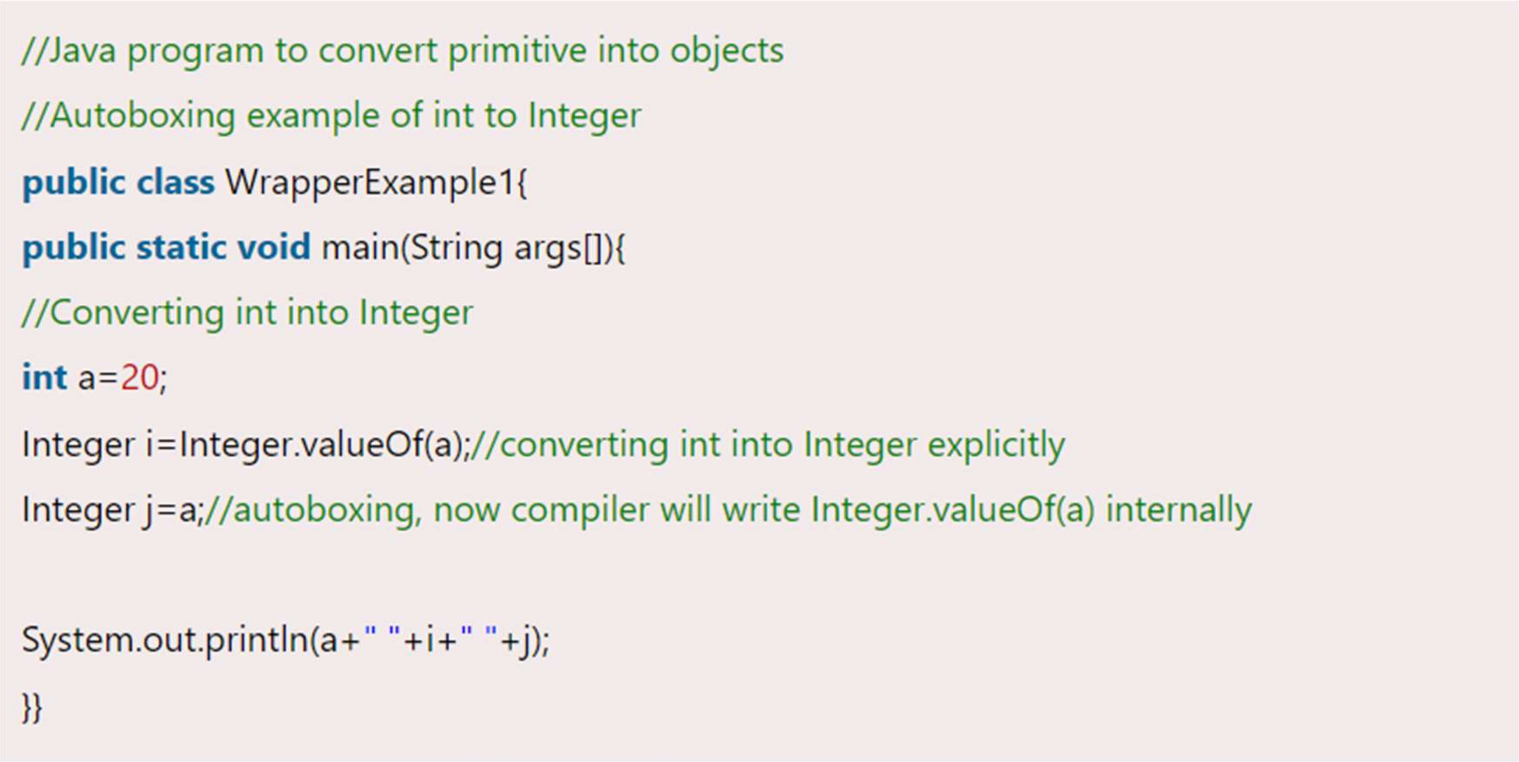


Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Why need Wrapper Classes

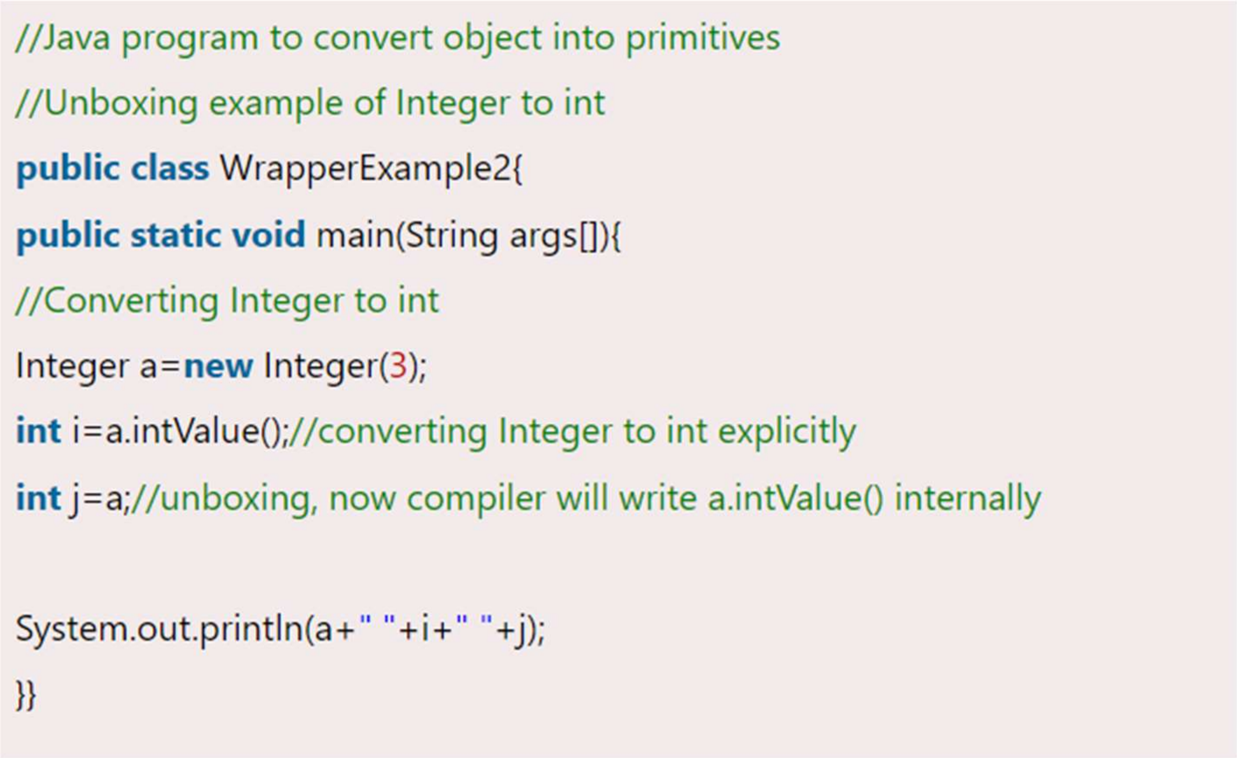
- Primitive types are not objects
  - With wrapper classes java can be completely object oriented
- For working with collections we need them, as primitive types can not store null values

Boxing and AutoBoxing



```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Unboxing



```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

Random

- A class in java.util
- Methods for generating random numbers
- nextInt()
- nextInt(int n): generates random number between 0 and n
- nextInt(-value): gives error

String vs StringBuffer vs StringBuilder

- String is immutable
- StringBuffer and StringBuilder are mutable
  - StringBuilder is faster
  - Both have a method called append

  - System.identityHashCode(s1)

Tasks

- Write a Java method to calculate the average of a variable number of doubles using varargs.
- Write a java program:
    - You will create a method which accepts variable number of integer arrays and print their elements