Classes and Objects: Intro

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Wednesday, 10 November 2021

Copyright © 2021

UNIVERSITY OF CALGARY

What is an object/class

- **Object**-oriented programming
  - **Objects** describe what is important in your application.
- In **Java**, a programmer describes what a **class** is (a template).
- A **class** describes a **set of objects with the same behavior**.
  - For example, the **String** class describes the behavior of all **strings**.
  - We call each individual **string** as an instance of the class **String**

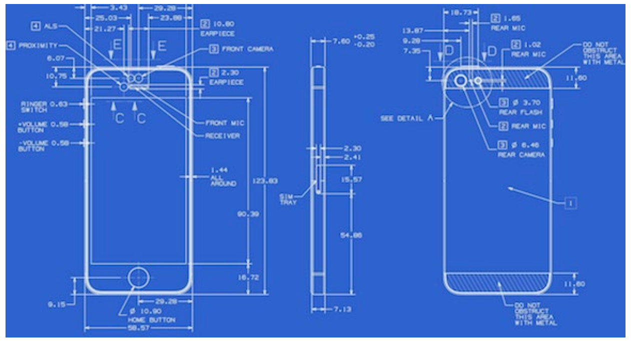
What is an object/class

- A **class** describes a **set of objects with the same behavior**.
  - For example, the **String** class describes the behavior of all **strings**.
  - We call each individual **string** as an instance of the class **String**
  - **Ex. Scanner scanner = new Scanner(System.in)**
  - **Scanner** is the **class description (and the type)**
  - **scanner** is an instance of **Scanner**

Class and Objects

Class

- A template that describes:
  - Fields (variables)
  - Methods (functions) operating on the data in the fields



Objects

- Instances of that class which take on different forms

Basics

UNIVERSITY OF CALGARY

Constructing an Object from a Class

- Variables of a class store pointers to objects (instances) of that class
- The process of creating an instance of an object is called instantiation/construction.
- Format:

  <name of the class> <object name> = new **<name of the class>** ()

- Example:

  Student student1 = new **Student**()

- The instantiation allocates memory space for the data fields and then associates the address with the object name

Static?

- Each class has **methods/fields** we can access
  - Methods are functions connected to a specific class

  - Methods/variables **without static** are **object methods/fields**
    - They are specific to internal data of each **instance** of the class

  - **static** methods/variables are **Class methods/ fields**
    - They are shared by all instances of that classes

Static?

- Each class has **methods/fields** we can access
  - Methods are functions connected to a specific class

  - Methods/variables **without static** are **object methods/fields**
    - They are specific to internal data of each **instance** of the class
    - THESE ARE NEW TO US
  - **static** methods/variables are **Class methods/fields**
    - They are shared by all instances of that classes
    - WE’VE ONLY BEEN CREATING THESE PREVIOUSLY
    - public static void main(String[] args) is an example of this

UNIVERSITY OF CALGARY

. (in context of)

- The **dot** tells Java we want to access a function/method of the particular object/class

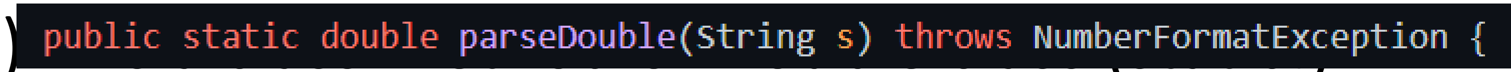
• Double.parseDouble(String s)	is a class method of Double class (static!)
• Double.NaN	is a class constant (static! and final!)
• scanner.nextLine()	is a object method for a scanner instance
• array.length	is an object constant (final!)

. (in context of)

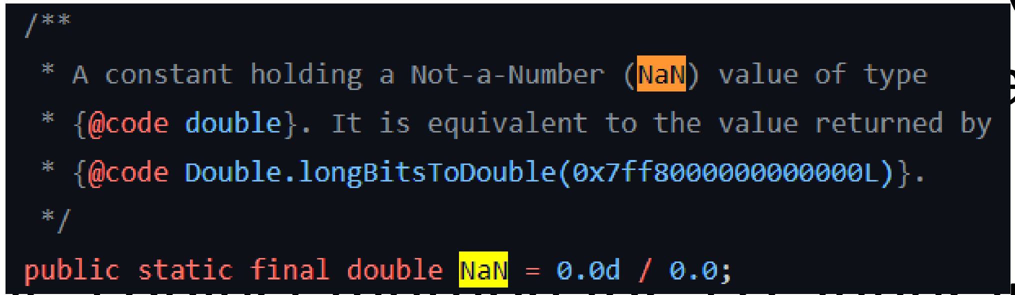
- The **dot** tells Java we want to access a function/method of the particular object/class

- Double.parseInteger(String s)
- Double.NaN al!)

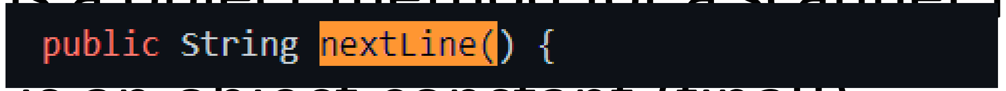
- scanner.nextLine() nstance
- array.length



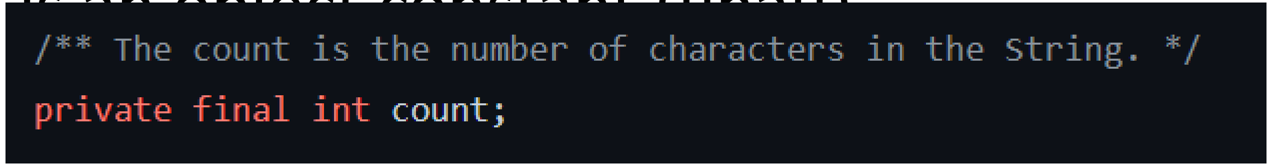
```
public static double parseDouble(String s) throws NumberFormatException {
```



```
/**
 * A constant holding a Not-a-Number (NaN) value of type
 * {@code double}. It is equivalent to the value returned by
 * {@code Double.longBitsToDouble(0x7ff8000000000000L)}.
 */
public static final double NaN = 0.0d / 0.0;
```



```
public String nextLine() {
```



```
/** The count is the number of characters in the String. */
private final int count;
```

Decisions in Object Design

1. **Encapsulation**
   - What is object representing? How is one object unique from another?
2. **Data**
   - Looking at what the object encapsulates, how do we capture that information. (vars)
3. **Methods**
   - How do we create a new object? (constructors)
   - What information about the object do we share? (private/public access)
   - How do we manipulate the information within the object? (accessors/mutators)
4. **Identity**
   - How can we tell if two instances of the objects are equal? (equals/compareTo)

UNIVERSITY OF CALGARY

Naming/Purpose

UNIVERSITY OF CALGARY

Choosing Classes

- A class represents a single concept from the problem domain
- Name for a class should be a noun that describes concept
- **Concepts from mathematics**:
  - Point
  - Rectangle
  - Ellipse
- **Concepts from real life**
  - BankAccount
  - CashRegister

Choosing Classes

- **Actors** (end in -er, -or)–objects do some kinds of work for you
  - Scanner
  - Random
- **Utility classes**–no objects, only static methods/constants (Helpers)
  - Math
- **Program starters**: only have a main method
- Don't turn actions into classes:
  - Paycheck is better name than ComputePaycheck

Create A Class

Let’s Create A Class

Tally counter –

What do we know about it?

Let’s Create A Class

Tally counter –

1. View count
2. Add 1 to count
3. Reset count to 0

I need to store some sort of integer data for the tally. Any other data?

UNIVERSITY OF CALGARY

Let’s Create A Class

What if we were using a tally counter in Java?

Let us make a new one

```
Counter tally = new Counter();
```

Now how would we use it

```
System.out.println(tally.getCount())
tally.count()
System.out.println(tally.getCount())
tally.reset()
System.out.println(tally.getCount())
```

UNIVERSITY OF CALGARY

Let’s Create A Class

```
public class Counter{
        //How do I create a counter?
        //Special function called Constructor
}
```

Let’s Create A Class

```
public class Counter{
        //How do I create a counter?
        public Counter(){
        }
}
```

Let’s Create A Class

How do we store data in classes?

```
public class Counter{
      static int var1;                    //Class variable
      int var2;                           // object/instance variable
      static final int VAR1 = 1;          //Class constant
      final int VAR2 = 2;                 //instance constant
                                                //(not overly useful versus class constant)
}
```

Let’s Create A Class

How do we store data in classes?

```
public class Counter{
    static int var1;            //shared for every Counter

        int var2;              // unique to each
                               // tally counter
}
```

Let’s Create A Class

```
public class Counter{
        int count;
        public Counter(){
        }
}
```

Instance variables that aren’t initialized in the constructor will default to Java’s default value (like arrays did).

So int will be count = 0;

Let’s Create A Class

```
public class Counter{
        int count = 0;
        public Counter(){
        }
}
```

```
public class Counter{
        int count;
        public Counter(){
                count = 0;
        }
}
```

```
public class Counter{
        int count;
        public Counter(){
                this.count =0;
        }
}
```

```
public class Counter{
        int count;  //default 0 will be assigned
        public Counter(){
        }
}
```

Object Scope!

this.

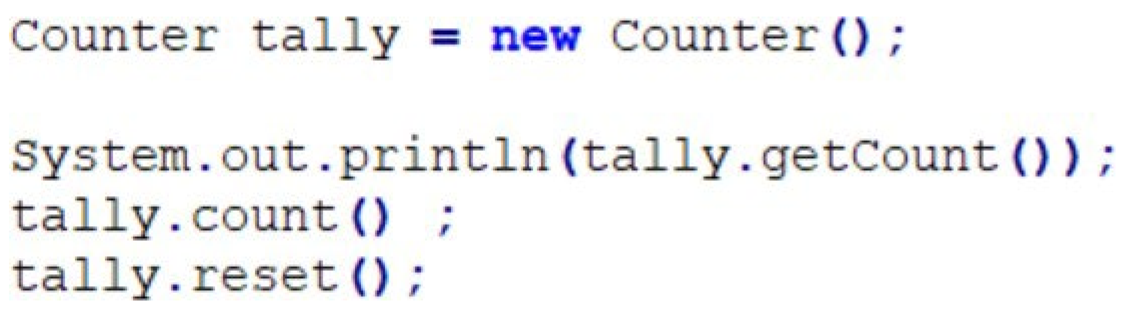
Tells Java we mean instance variable

Technically unnecessary unless we have used the same name for other function variables

```
public void setCount(int count){
    count = count; //BROKEN
}

public void setCount(int count){
    this.count = count;
}
```

Let’s Create A Class



```
Counter tally = new Counter();

System.out.println(tally.getCount());
tally.count() ;
tally.reset();
```

For these method calls **tally** is known as the **implicit parameter**

While any parameters passed inside the ellipses (…) are **explicit parameters**

Whatever object **instance** was **tally.** when the call was made becomes referenced by **this.** Inside the class method definitions

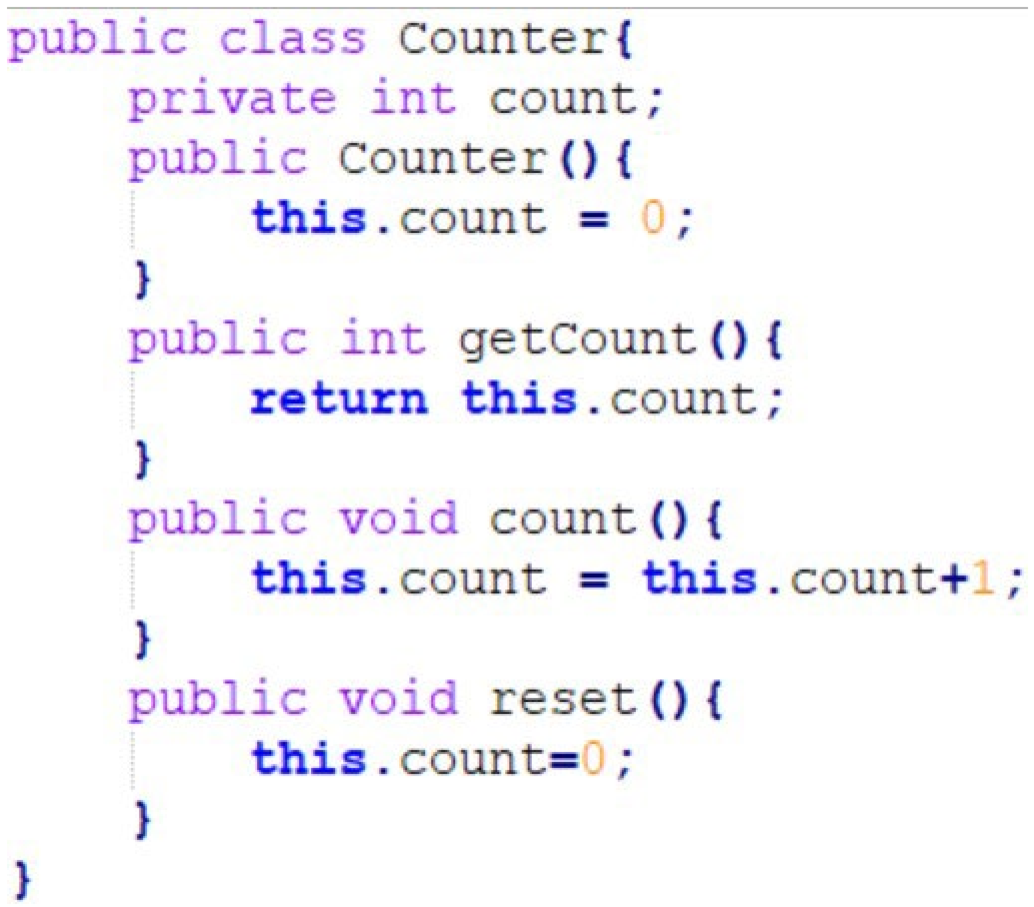
Let’s Create A Class

```
public class Counter{
        int count;
        public Counter(){
                this.count = 0;
        }
    //View count
    //Add 1 to count
    //Reset count to 0
}
```

Let’s Create A Class

```
public class Counter{
    private int count;
    public Counter(){
        this.count = 0;
    }
    public int getCount(){
        return this.count;
    }
    public void count(){
        this.count = this.count+1;
    }
    public void reset(){
        this.count=0;
    }
}
```

Let’s Create A Class



```
public class Counter{
    private int count;
    public Counter(){
        this.count = 0;
    }
    public int getCount(){
        return this.count;
    }
    public void count(){
        this.count = this.count+1;
    }
    public void reset(){
        this.count=0;
    }
}
```

For java methods/variables

- **public** – any other code can access
- **private** – only internal class access

count is an **instance variable**

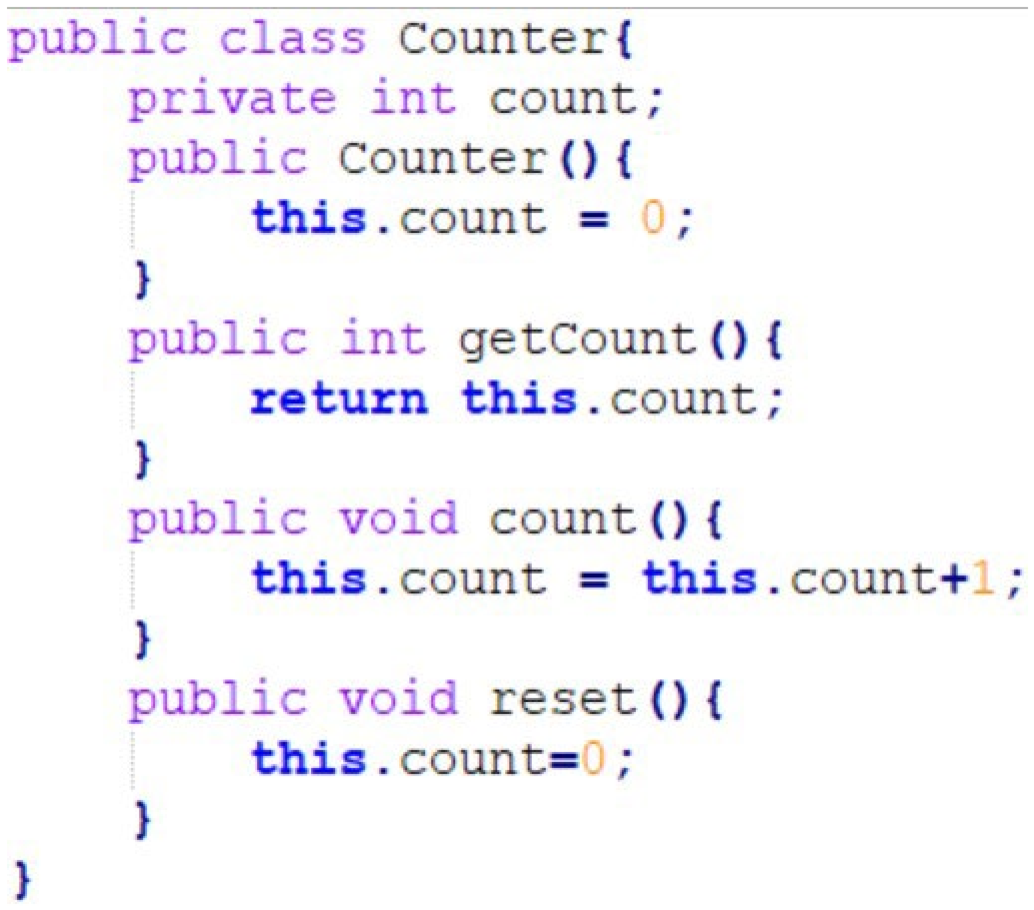
- one count var exists for each new Counter()

Instance variables should be private

- Access/Modification via instance methods

Most instance methods are public (unless they shouldn’t be used externally)

Let’s Create A Class



```
public class Counter{
    private int count;
    public Counter(){
        this.count = 0;
    }
    public int getCount(){
        return this.count;
    }
    public void count(){
        this.count = this.count+1;
    }
    public void reset(){
        this.count=0;
    }
}
```

When private

Counter tally = new Counter();

tally.count();

tally.getCount();   //Gives us 1

If count was public?

Counter tally = new Counter();

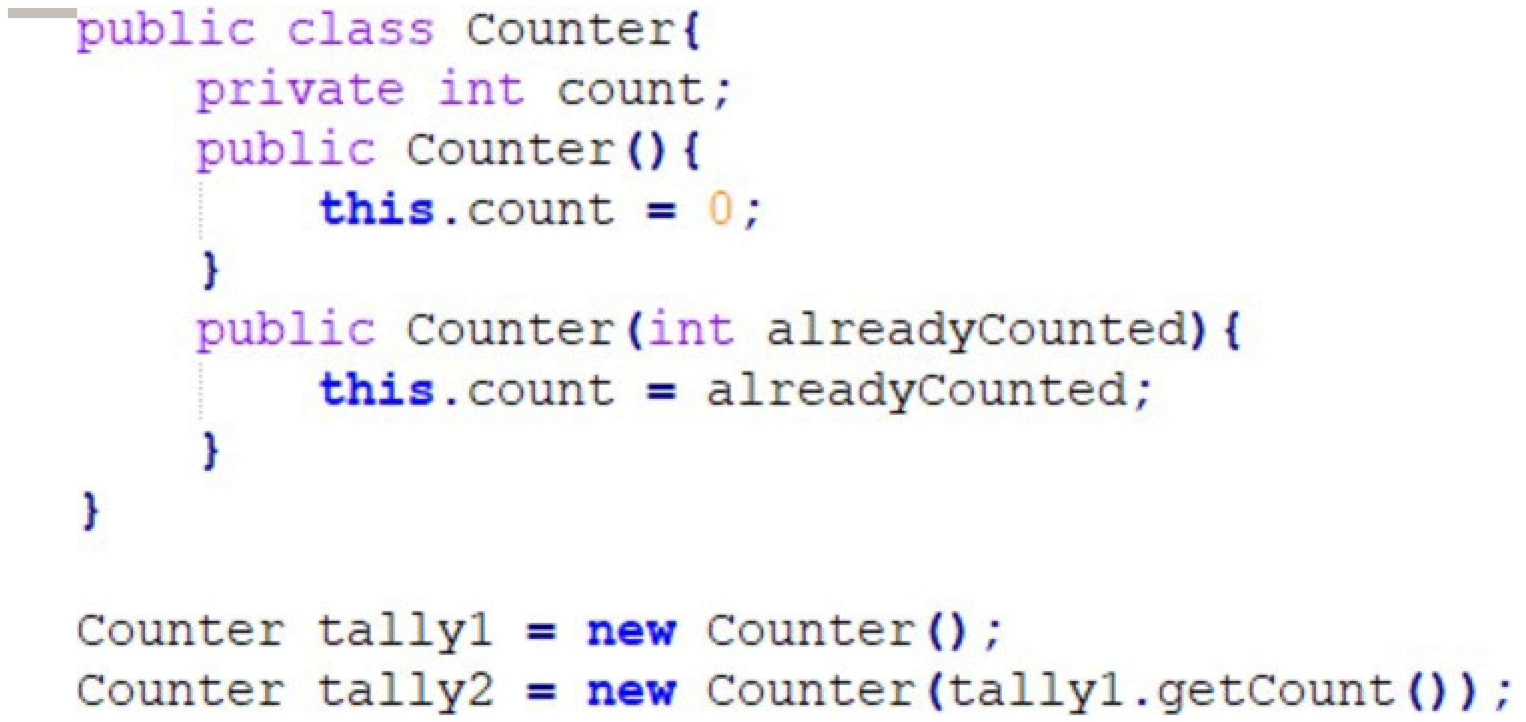
tally.count = 500;   //allowed now

tally.count();

tally.getCount();   //Gives us 501

Constructors

Constructor Overloading



```
public class Counter{
    private int count;
    public Counter(){
        this.count = 0;
    }
    public Counter(int alreadyCounted){
        this.count = alreadyCounted;
    }
}

Counter tally1 = new Counter();
Counter tally2 = new Counter(tally1.getCount());
```

We can **overload** a constructor (same name, different parameters)

We now can make a new object through different means

Constructors

If you do not initialize an instance variable in a constructor
it is automatically set to a default value:

- Numbers are set to zero. (base types, not Objects)
- Boolean variables are initialized as false. (base types, not Objects)
- Object and array references are set to the special value **null** that indicates that no object is associated with the variable.
    - This is often not desirable

Informative Printing

UNIVERSITY OF CALGARY

Printing

Every object shares as a base starting point

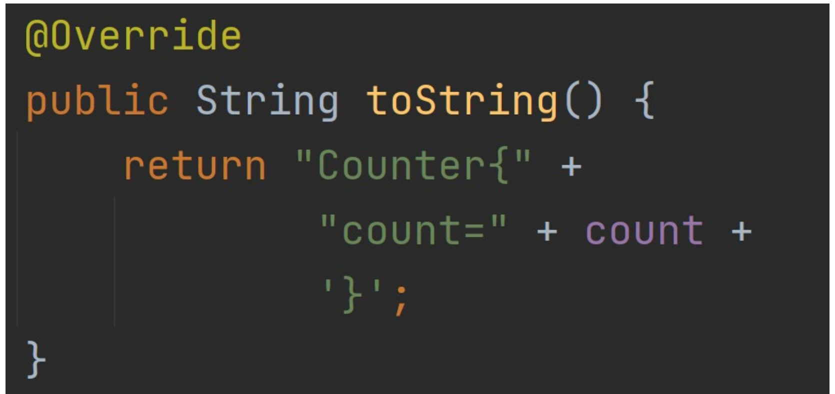
https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Which has a instance method **public String toString()**

Which has a default print for every object

getClass().getName() + '@' + Integer.toHexString(hashCode())

We can replace this with our own String (@Override is recommended)



```
@Override
public String toString() {
    return "Counter{" +
            "count=" + count +
            '}';
}
```

UNIVERSITY OF CALGARY

Public Interface

UNIVERSITY OF CALGARY

Public Interface

What developers can see about your class

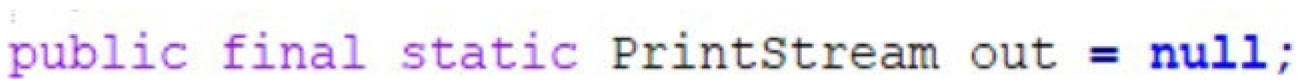
often packaged up as API (Javadoc)

- Public variables/constants
- Public Constructors
- Public Accessors
- Public Mutators

Expose only what is necessary

Public Interface

java.lang includes System.java where System.out is a PrintStream

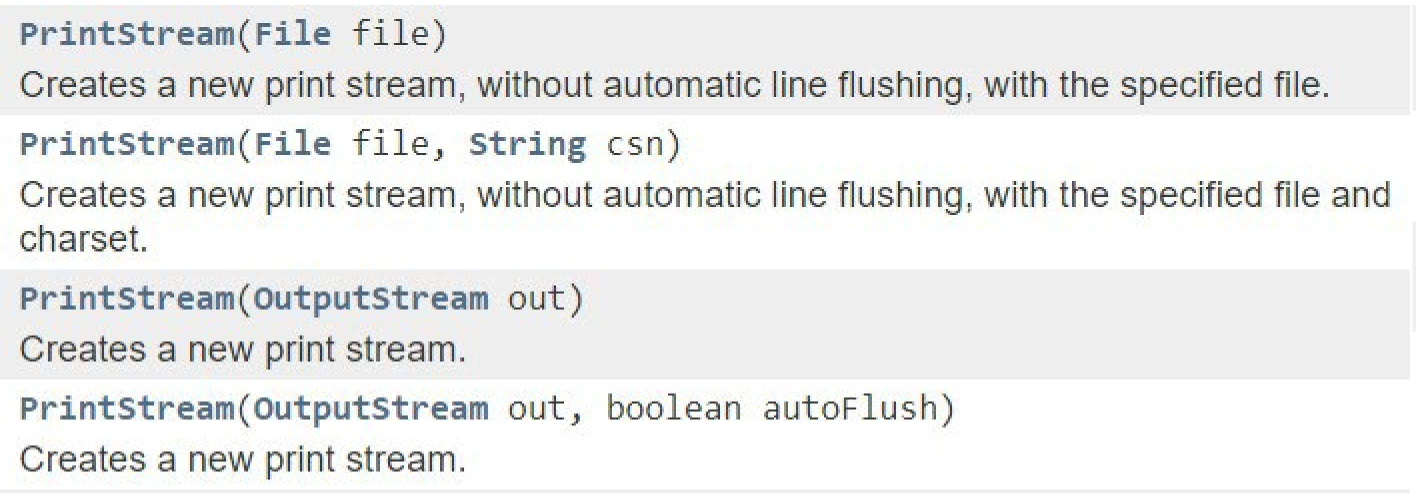


```
public final static PrintStream out = null;
```

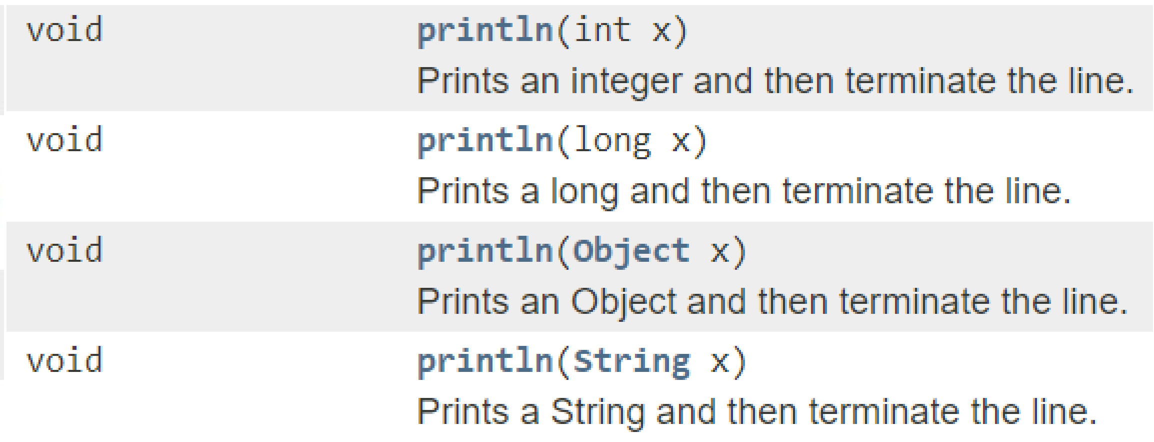
We can look at the **public interface** for PrintStream at

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/PrintStream.html

The internal details may be unknown **(private implementation)**



`PrintStream(File file)` Creates a new print stream, without automatic line flushing, with the specified file.
`PrintStream(File file, String csn)` Creates a new print stream, without automatic line flushing, with the specified file and charset.
`PrintStream(OutputStream out)` Creates a new print stream.
`PrintStream(OutputStream out, boolean autoFlush)` Creates a new print stream.



`void`	`println(int x)` Prints an integer and then terminate the line.
`void`	`println(long x)` Prints a long and then terminate the line.
`void`	`println(Object x)` Prints an Object and then terminate the line.
`void`	`println(String x)` Prints a String and then terminate the line.

Onward to … Design

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY