Warmup

```
// This program has at least 5 errors.
// Work with the person next2U to find them all!

public class Errors {


      public static void main(String[] args)
            String temperature = 80.3;
            int n = 100
            n = "Wait, what?";
            print("This is fine.");
}
```

Class norms

This course will strive to create an inclusive learning environment in which everyone feels like they belong in the class. The following norms are designed to help us collectively reach that ideal. Please make a good effort to live out these norms throughout the semester.

- Listen with the possibility of being changed.
  Speak with the promise of being heard.
- Be present and be your best self.
- Everyone has something to learn. No one person is good at everything or has all the skills to complete a group-worthy task.
- Everyone has expertise to offer. Every person has relevant strengths to bring to each group-worthy task.
- We need each and every person in this group.
- You have the right to ask for help, and the duty to assist.
- Be willing to experience discomfort.
- Expect and accept non-closure.

Java Constructs

Variables and their Types

Assignments

Operators (relational, logical)

Conditionals, boolean expressions

Loops via iteration (while, for, do)

Control flow and memory model

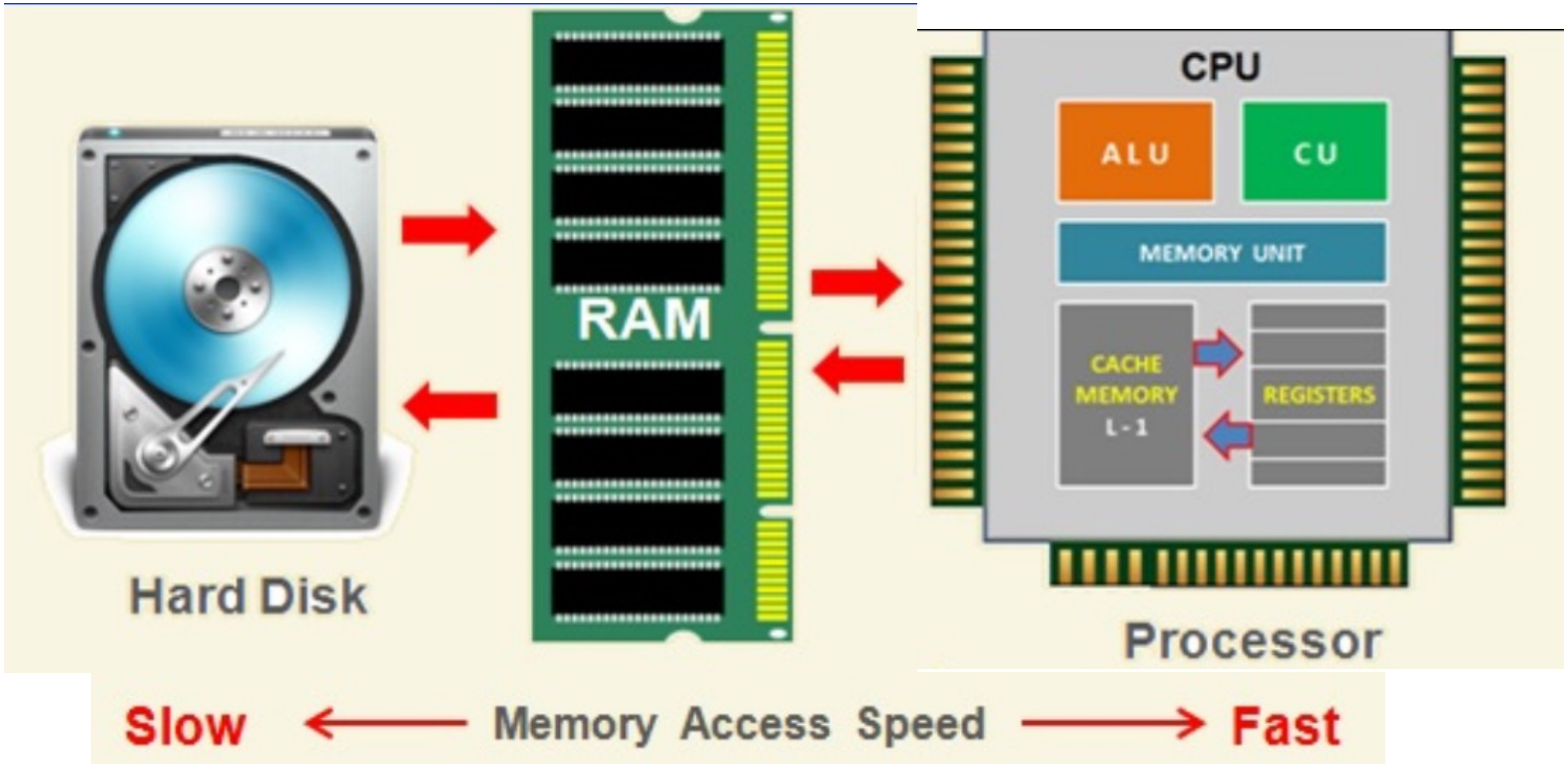
- Code is executed sequentially altering memory contents, e.g.

```
1. int x;
2. int y;
3. int z;
4. x = 7;
5. y = 5;
6. z = x + y;
7. System.out.println(z);
```

Line number	Line 1	Line 2	Line 3	Line 4	Line 5	Line 6	Line 7
Memory model of the current state of the program	x (int)	x (int) y (int)	x (int) y (int) z (int)	x (int) 7 y (int) z (int)	x (int) 7 y (int) 5 z (int)	x (int) 7 y (int) 5 z (int) 12	x (int) 7 y (int) 5 z (int) 12
Other actions							Print value of z (12) to screen

Accessing variables

Data may or may not be in memory, and this can affect execution



Like every programming language, Java has...

- Variable declaration
  - `int x;`
- Assignment statement ("gets")
  - `x = 5; double pi = 3.14;`
- Conditional statements
  - ```
    if (x == 5) {
            x = x+1;
    } //We don't take fives!
    ```
- Loops
  - ```
    while (x >= 5) {
        x = x-1;
    }
    ```
- Functions (aka: Methods)
  - ```
    public static int increment(int x) {
        return x+1; }
    ```


    ```

Relational Operators

- Java has a **`boolean`** type that can take the value `true` or `false`
- **`boolean b = (x < 5); // parentheses are optional here`**
- Booleans arise naturally when using **relational operators** to compare two values

3 < 5	
3 < 2	
3 > 2	
5 <= 1	
5 >= 1	
5 == 5	
5 == 6	
5 != 6	

Logical Operators

- Boolean values can be manipulated with the logical operators **!** (not), **&&** (and), and **||** (or)

! (3 < 5)	
! (3 == 5)	
(3 > 5) && (7 < 8)	
(3 < 5) && (7 < 8)	
(3 > 5) || (7 < 8)	
(3 > 5) || (7 > 8)	

Predicate methods (returning T/F)

- A **predicate** is any method that returns a `boolean` value

```
//determine if n is even
public static boolean isEven(int n){
        return (n % 2) == 0;
}
//determine if num is divisble by factor
public static boolean isDivisibleBy(int num, int factor) {

}
//determine if n is between lo and hi
public static boolean isBetween(double n, double lo, double hi)
{

}
```

Write your own Predicate to determine if n is odd

```
public static boolean isOdd(int n){

}
```

Then, can you write it another way using another predicate?

```
public static boolean isOdd(int n){

}
```

Conditionals

- To control the program flow and choose between two courses of action, we use **conditional statements** such as: `if`, `else if`, and `else`

```
//returns absolute value of n
public static double abs(double n){
        if (n < 0) {
                return -n;
        } else {
                return n;
        }
}
//returns absolute value of n (in a little surprising way)
public static double abs(double n){

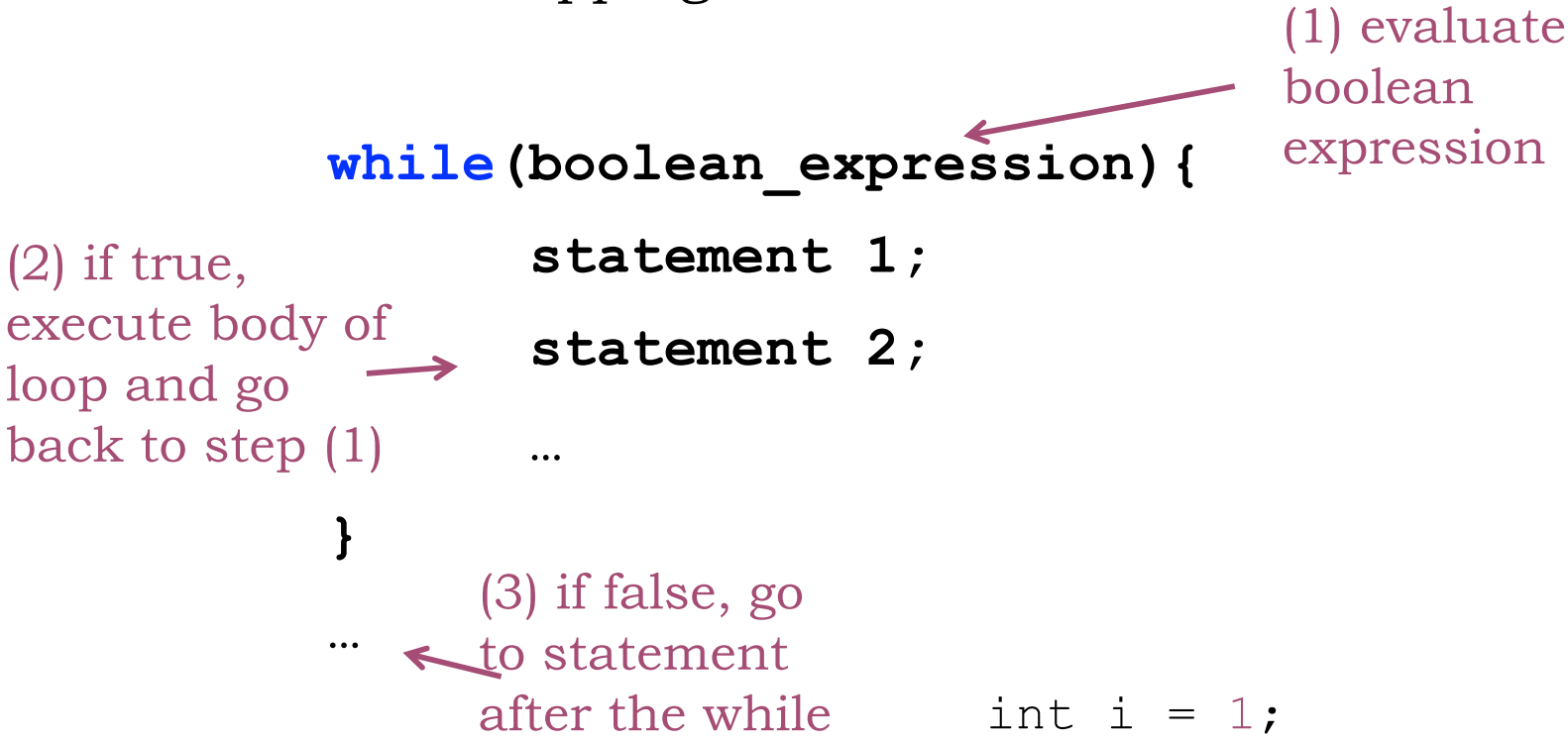
}
```

Be the computer: Run by hand. What does it print?

```
public static void main(String[] args){
        int x = __; String s = "meow";
        if(x < 30 && s.length() < 10){
                x = x + 5;
                int y = s.length();
                if(x+y > 36){
                        System.out.println("hello " + x);
                }else if(x+y < 33){
                        System.out.println( "howdy " + y);
                }else{
                        System.out.println("hi!");
                }
        } else {
                x = x - 10;
                int y = s.length() + 5;
                if( x == 15) System.out.println( "Salut " + x);
                else System.out.println("Ciao " + y);
        }
}
```

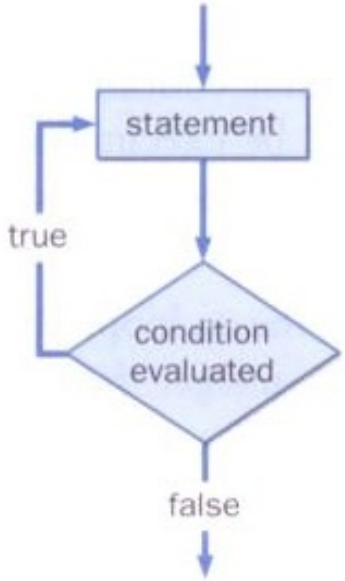
Repetition through Iteration – while loop

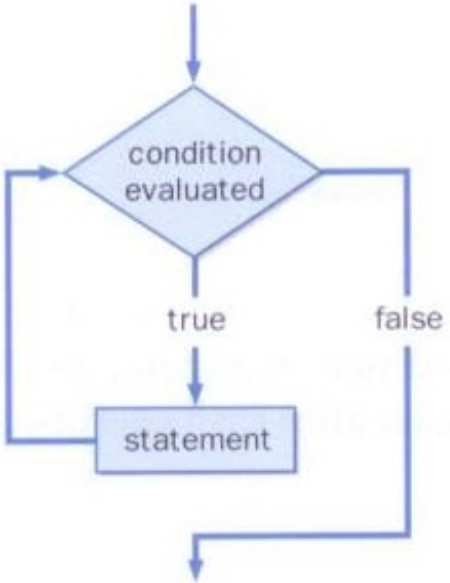
- **Iteration** refers to a sequence of steps that is repeated until some stopping condition is reached

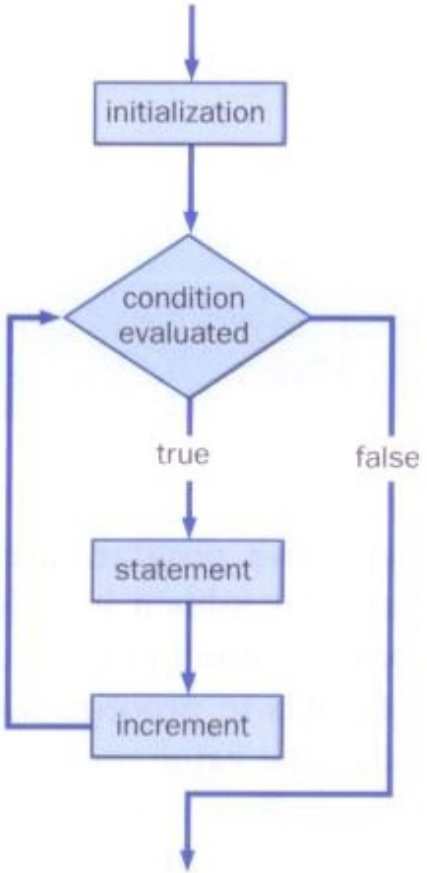



```
int i = 1;
while (i < 4) {
    System.out.println("CS230");
    i = i + 1;
}
```

Java has several loop statements: Do-loop vs While-loop vs For loop

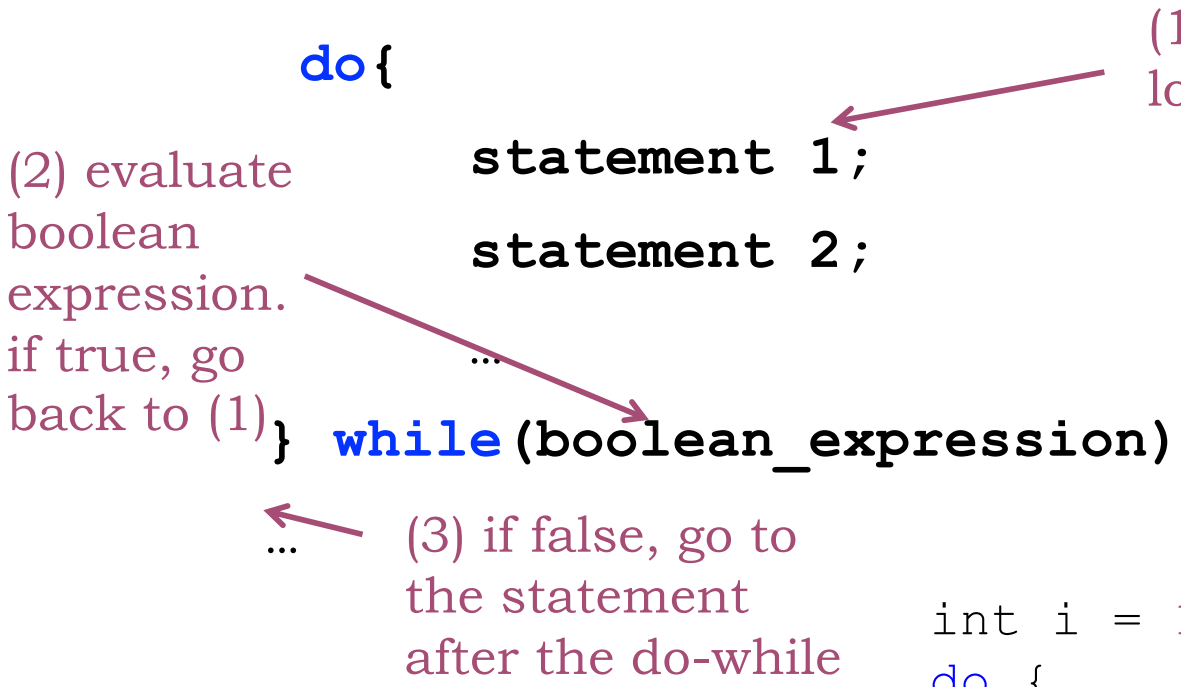






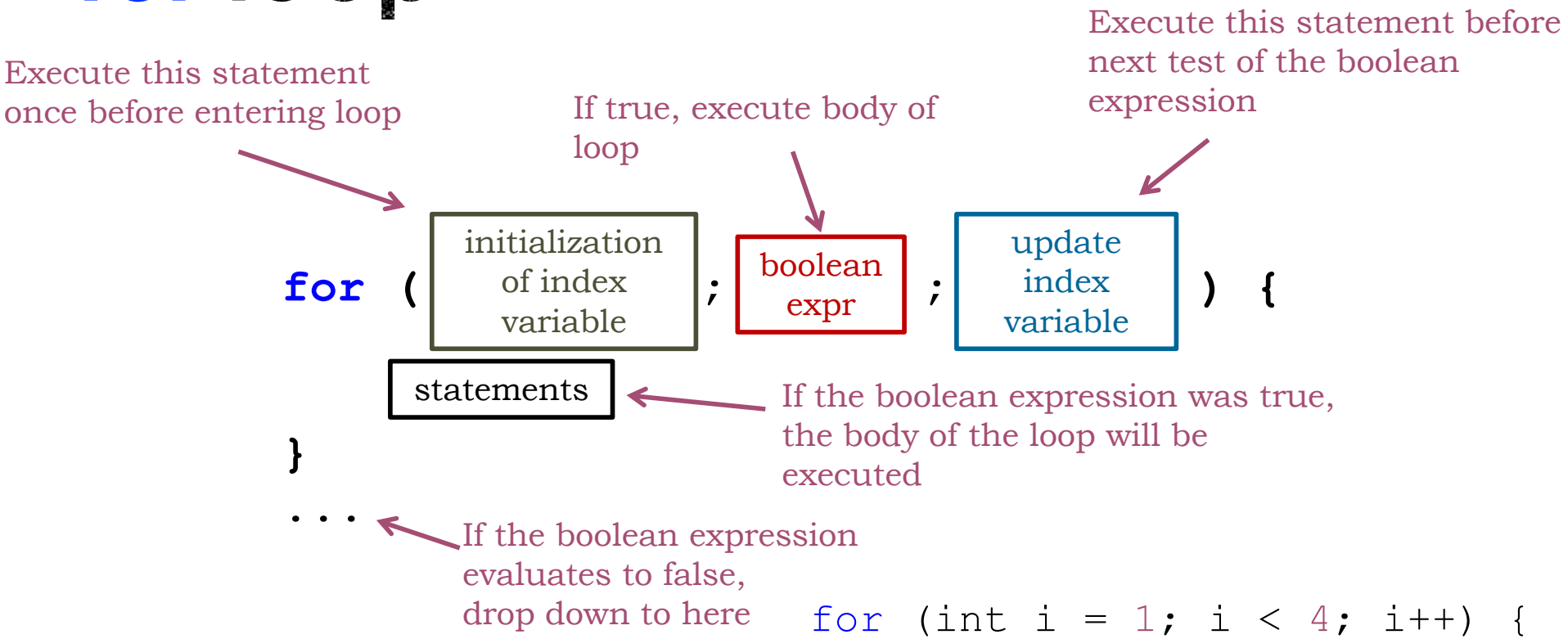
Repetition through Iteration – do loop

- **Iteration** refers to a sequence of steps that is repeated until some stopping condition is reached



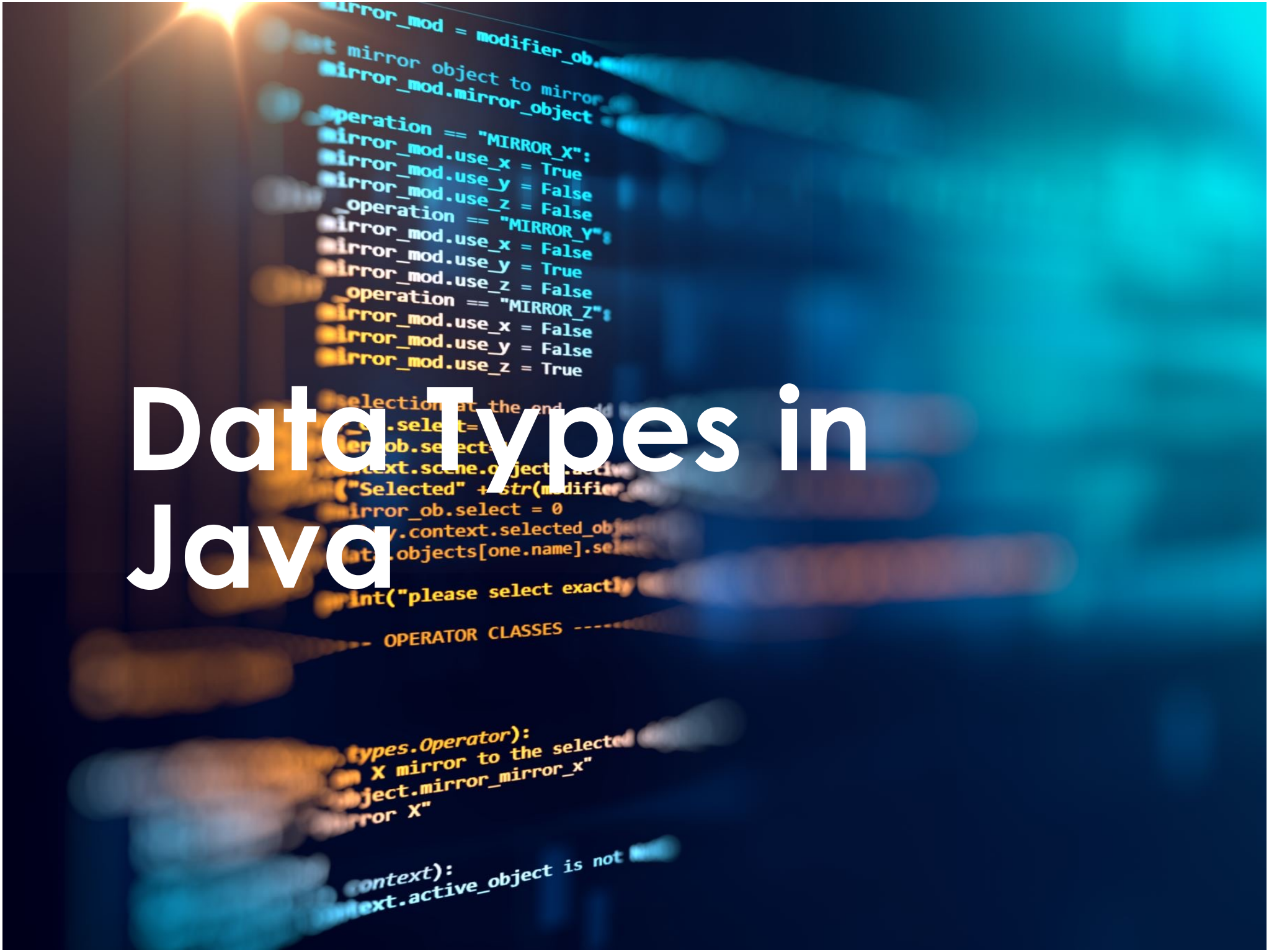

```
int i = 1;
do {
     System.out.println("CS230");
     i = i + 1;
} while (i < 4);
```

Repetition through Iteration – for loop




```
for (int i = 1; i < 4; i++) {
    System.out.println("CS230");
}
```

Write a for loop that prints the numbers from 1 to 10.


Data Types in Java

Data Types in Java

- Java is a **statically typed** language
  - You must define the type of each variable when it is declared
- Unlike Python, not all variables in Java are objects
  - Some are **primitive data types (**but have related objects)

Primitive	Storage	Range of values
int	32 bits	-2,147,483,648 to 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	32 bits	Approx. –3.4E+38 to +3.4E+38 with 7 significant digits
double	64 bits	Approx. –1.7E+308 to +1.7E308 with 15 significant digits
char	16 bits	65,535 Unicode characters
boolean	1 bit	true or false

Decimal Numbers

```
double num = 5.2;
num = 1.4;
num = num * 2.0;
System.out.println(num);
```

```
double fahrenheit = 98.6;
double celsius = (fahrenheit - 32) * 5 / 9;
System.out.println(celsius);
```

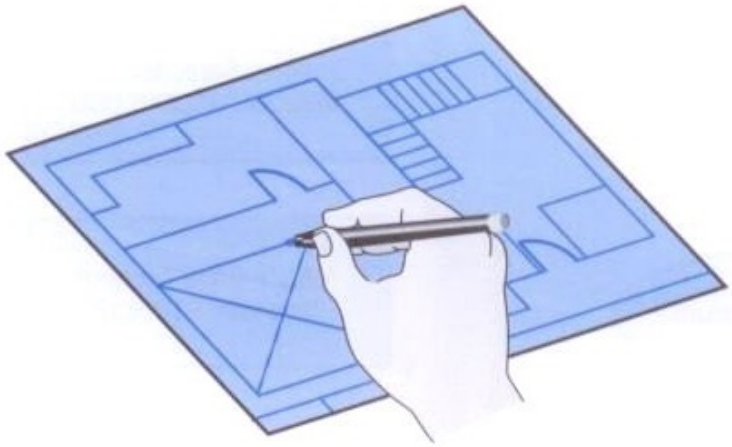
Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (`float` or `double`)
- Two floating point values are equal only if their underlying binary representations match exactly
- Computations often result in slight differences that may be irrelevant
- In many situations, you might consider two floating point numbers to be “close enough” even if they aren't exactly equal


Using Objects
(and Strings in particular)

OOP languages have Classes Classes create Objects

- A class is like a house blueprint from which you can create many of the "same" type of house
  (maybe with different characteristics)



Creating String Objects

- Generally, we use the `new` operator to create an object:

```
String title = new String("Hello CS230!");
```

This calls the String *constructor*, which is a special method that sets up the object

- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

Creating String Objects

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java rocks!";
```

- This is special syntax that works only for strings
- Each string literal (enclosed in double quotes) represents a `String` object

Invoking Methods

- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
int count = title.length()

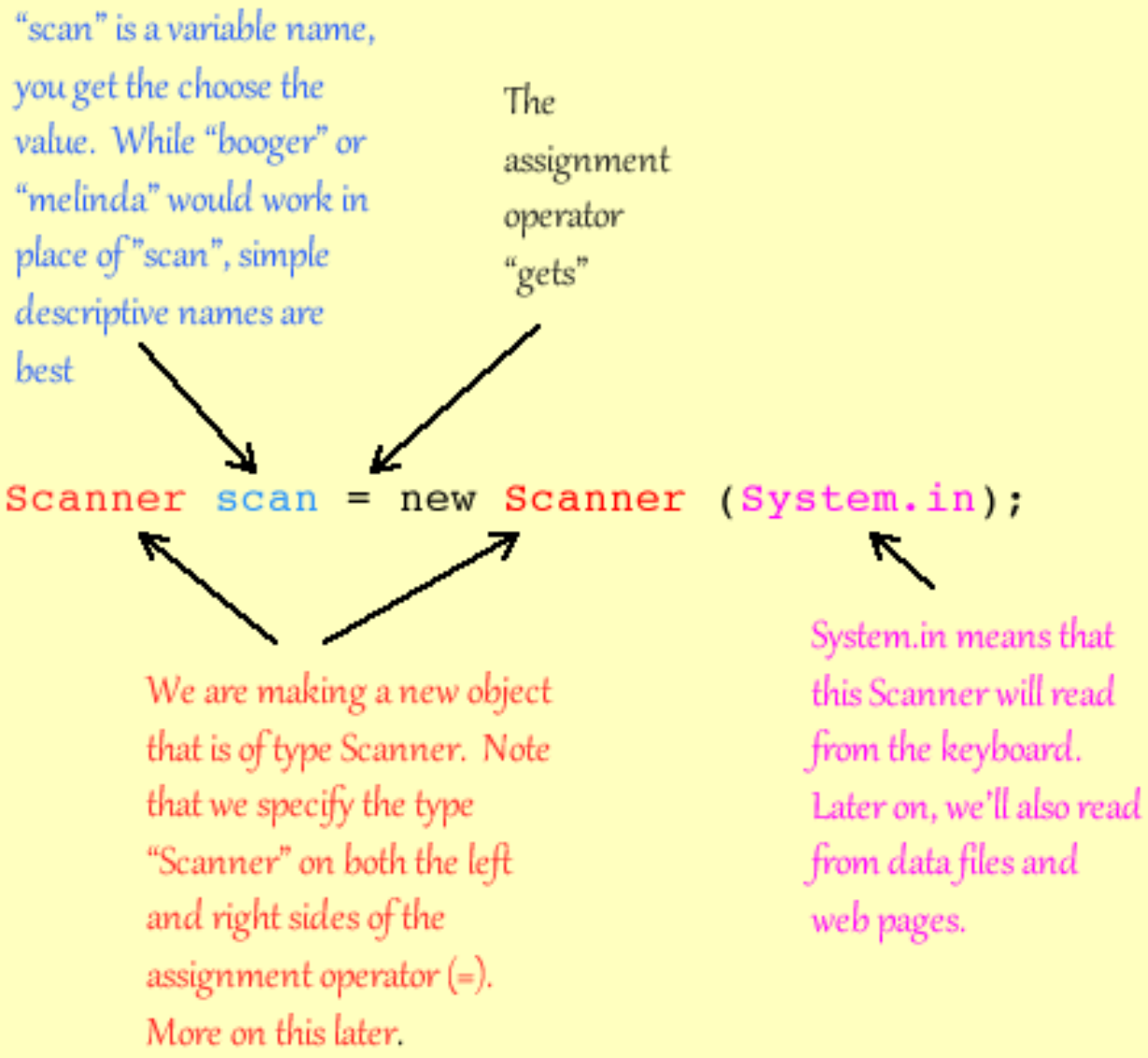
String line = scan.nextLine();
 //handy on the next assignment
```

- A method may *return a value*, which can be used in an assignment or an expression
- A method invocation can be thought of as asking an object to perform a service

Enough for a day…

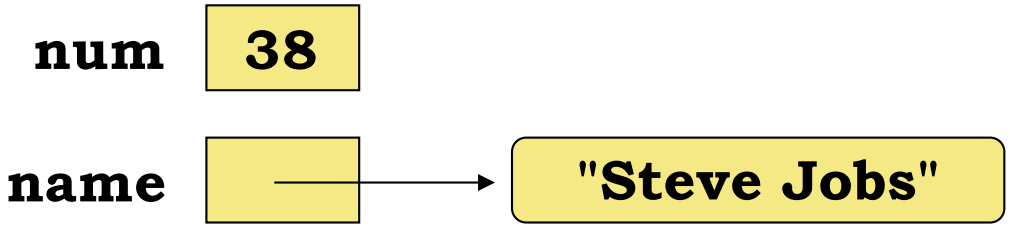
Declaring and initializing a new Scanner object

Useful in getting input from the user



Object References

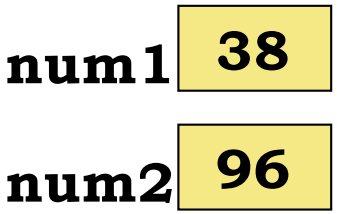
- While a primitive variable contains the value itself…
  - `int num = 38;`
- An object variable contains the ***address of the object***
  - `String name = "Steve Jobs"`
- An object reference can be thought of as a pointer to the location of the object
- Rather than dealing with arbitrary addresses, we often depict a reference graphically



Assignment Revisited

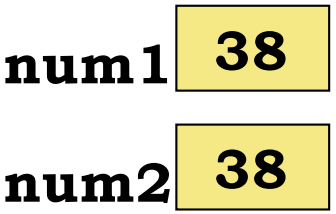
- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

Before:




```
num2 = num1;
```

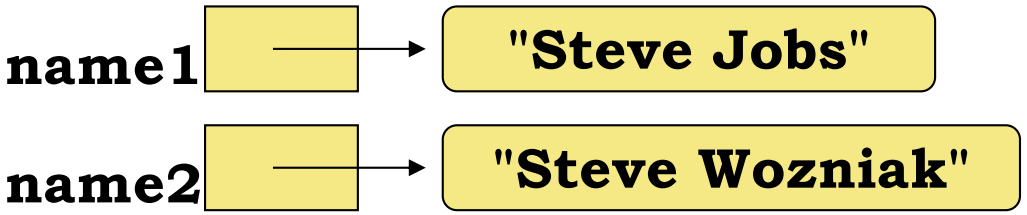
After:



Assignment Revisited

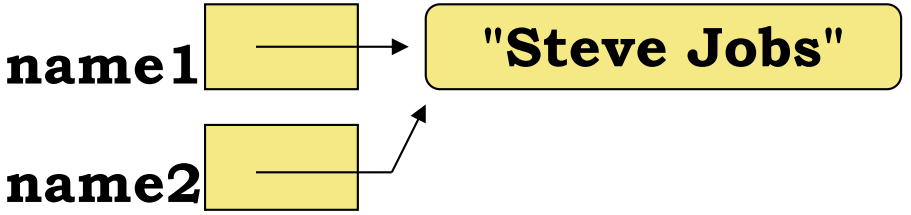
- For object references, only the address is copied (not the value) :

Before:




```
name2 = name1;
```

After:

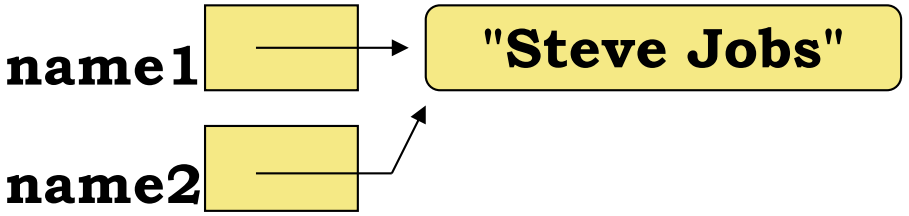


Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Check

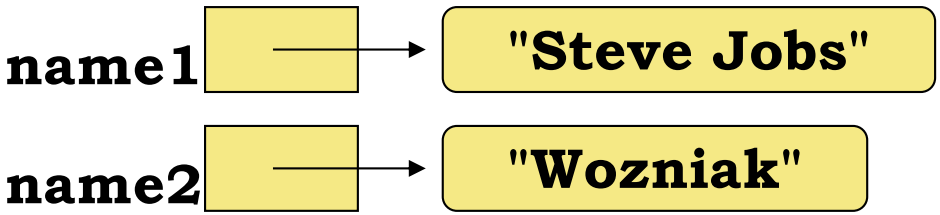
Given:



- What do you think happens if you execute

```
name2 = "Wozniak";
```

Result:



Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program
- The object is useless, and therefore is called *garbage*
- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection explicitly

The String Class

- Probably the most used class in Java
- Once a `String` object has been created, neither its value nor its length can be changed
- Thus we say that an object of the `String` class is *immutable*
- However, several methods of the `String` class return new `String` objects that are modified versions of the original

The String Class

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at 0 in each string
- In the string `"Hello"`, the character `'H'` is at index 0 and the `'e'` is at index 1
- What characters are located at:
  - `"Hello".charAt(0)`
  - `"Hello".charAt(4)`

- Some methods of the `String` class:

`String (String str)`
Constructor: creates a new string object with the same characters as `str`.

`char charAt (int index)`
Returns the character at the specified `index`.

`int compareTo (String str)`
Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value), the string `str`.

`String concat (String str)`
Returns a new string consisting of this string concatenated with `str`.

`boolean equals (String str)`
Returns true if this string contains the same characters as `str` (including case) and false otherwise.

`boolean equalsIgnoreCase (String str)`
Returns true if this string contains the same characters as `str` (without regard to case) and false otherwise.

`int length ()`
Returns the number of characters in this string.

`String replace (char oldChar, char newChar)`
Returns a new string that is identical with this string except that every occurrence of `oldChar` is replaced by `newChar`.

`String substring (int offset, int endIndex)`
Returns a new string that is a subset of this string starting at index `offset` and extending through `endIndex-1`.

`String toLowerCase ()`
Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.

`String toUpperCase ()`
Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.

Strings in Java vs Python

- Strings in Java and Python are quite similar.
  - Like with Python, Java strings are immutable.
- The difference is that to process Strings,
  Java uses method calls where Python uses Operators.

Python	**Java**	**Description**
str[3]	str.charAt(3)	Return character in 3rd position
str[2:5]	str.substring(2,4)	Return substring from 2nd to 4th
len(str)	str.length()	Return the length of the string
str.find('x')	str.indexOf('x')	Find the first occurrence of x
str.split()	str.split('\s')	Split the string on whitespace into a list/array of strings
str.split(',')	str.split(',')	Split the string at ',' into a list/array of strings
str + str	str.concat(str)	Concatenate two strings together
str.strip()	str.trim()	Remove any whitespace at the beginning or end

```
String s1 = new String("Grace Hopper");
String s2 = "CU L8R";
String s3 = ":)";
System.out.println(s1.toLowerCase());

System.out.println(s1.length());

System.out.println(s2.length());

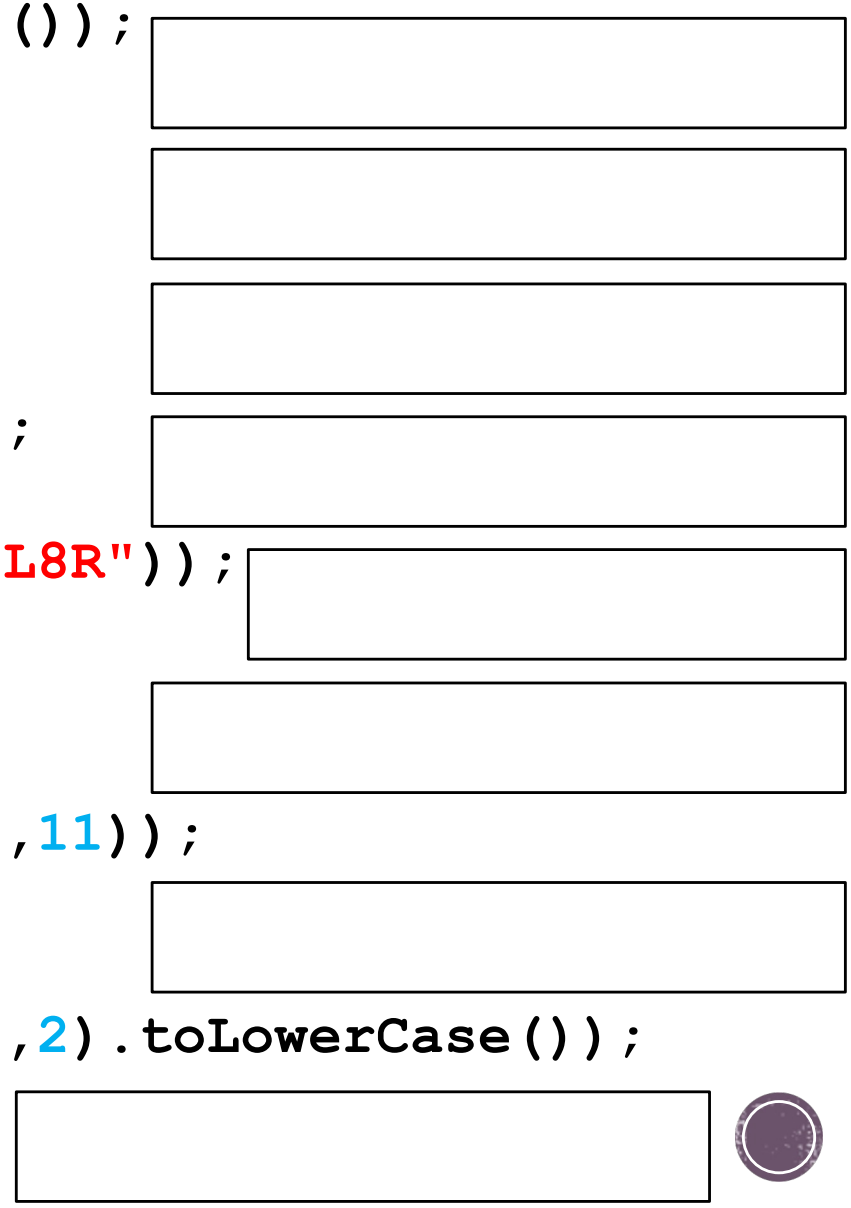
System.out.println(s2.equals(s3));

System.out.println(s2.equals("CU L8R"));

System.out.println(s2.charAt(1));

System.out.println(s1.substring(7,11));


System.out.println(s2.substring(0,2).toLowerCase());
```



Count Vowels

```
// Returns true if character is lower-case
// vowel (a, e, i, o, u), false otherwise.
public static boolean isVowel(char ch){



}
// Returns the number of vowels in the String s
public static int countVowels(String s){



}
```


The Java API

The Java API

- A *class library* is a collection of classes that we can use when developing programs
- The *Java API* is the standard class library that is part of any Java development environment
- API stands for Application Programming Interface
- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java API
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java API are organized into *packages*

Package	Provides support to
`java.applet`	Create programs (applets) that are easily transported across the Web.
`java.awt`	Draw graphics and create graphical user interfaces; AWT stands for Abstract Windowing Toolkit.
`java.beans`	Define software components that can be easily combined into applications.
`java.io`	Perform a wide variety of input and output functions.
`java.lang`	General support; it is automatically imported into all Java programs.
`java.math`	Perform calculations with arbitrarily high precision.
`java.net`	Communicate across a network.
`java.rmi`	Create programs that can be distributed across multiple computers; RMI stands for Remote Method Invocation.
`java.security`	Enforce security restrictions.
`java.sql`	Interact with databases; SQL stands for Structured Query Language.
`java.text`	Format text for output.
`java.util`	General utilities.
`javax.swing`	Create graphical user interfaces with components that extend the AWT capabilities.
`javax.xml.parsers`	Process XML documents; XML stands for eXtensible Markup Language.

Import Declarations

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

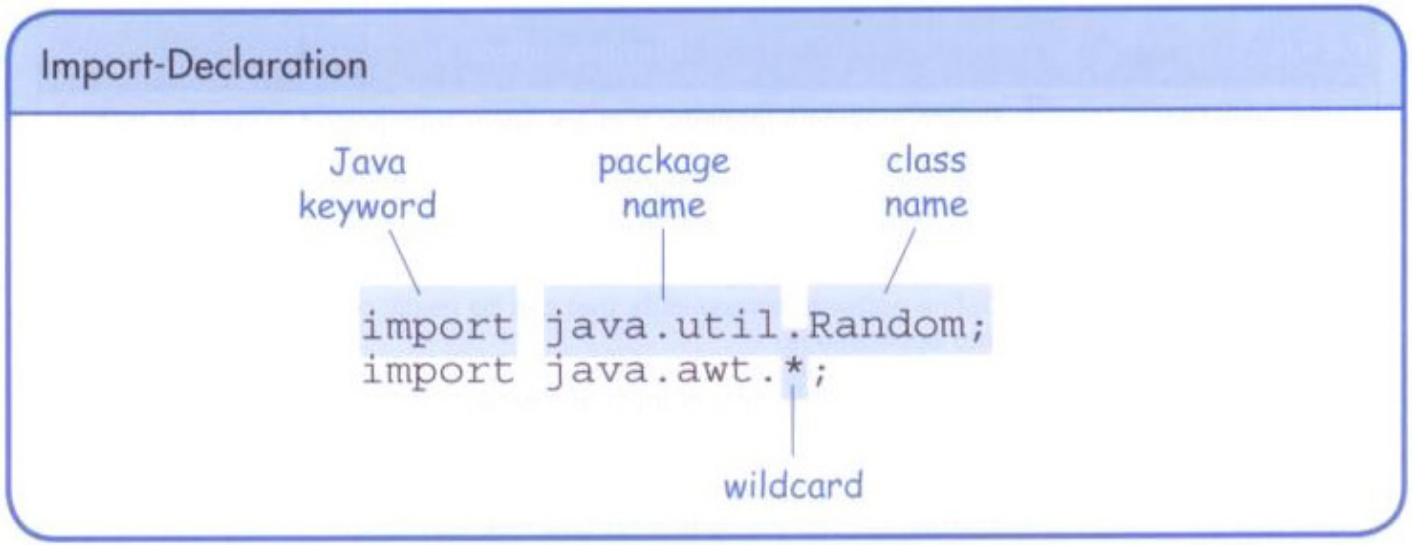
- Or you can *import* the class, and then use just the class name:

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character:

```
import java.util.*;
```

Import Declarations

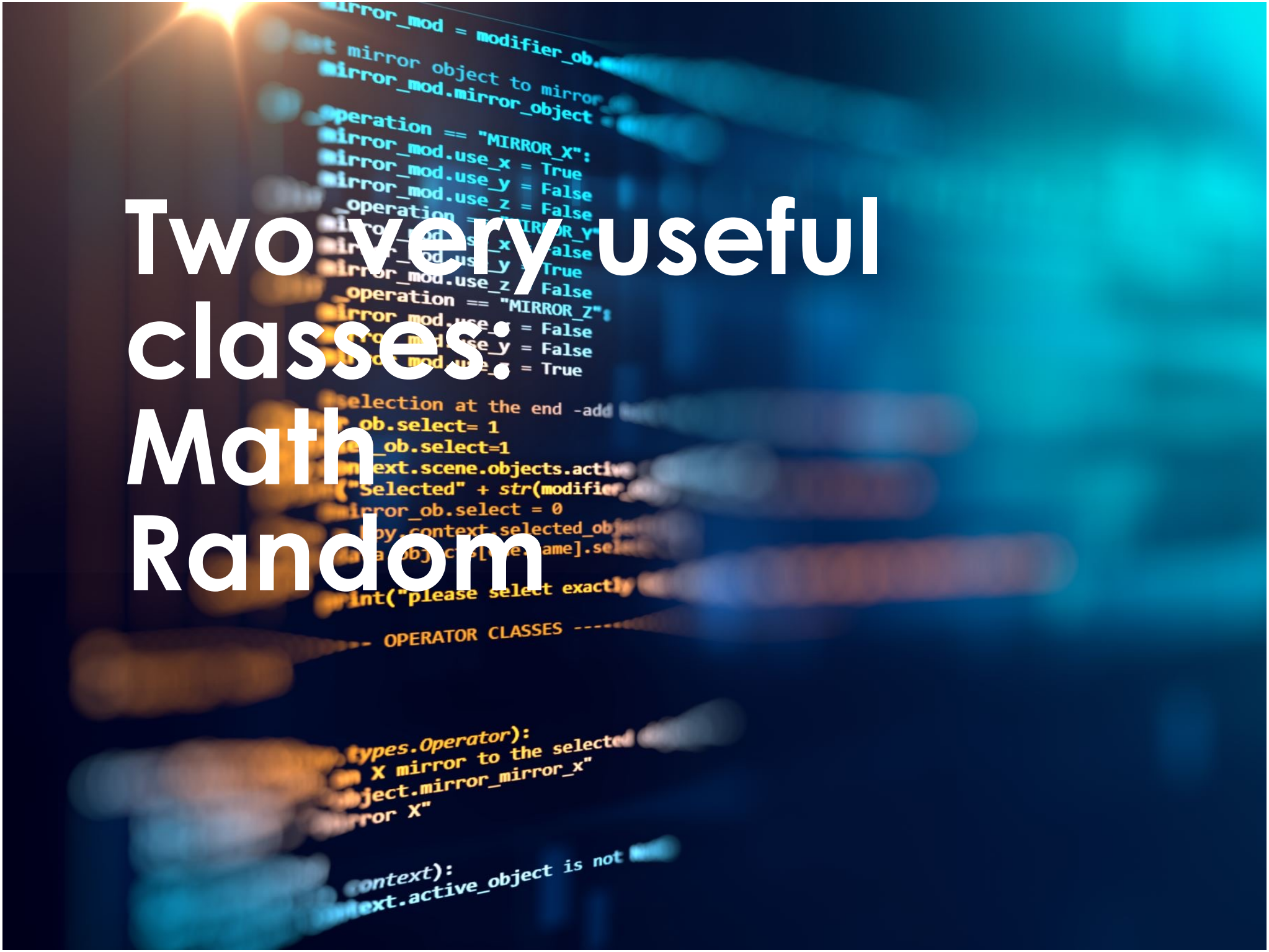


The java.lang Package

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line

```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported


Two very useful classes: Math Random

Math Class

```
System.out.println(Math.max(100, 50));
System.out.println(Math.sqrt(25));
System.out.println(Math.log(10));
```

```
// Given area of circle, returns the circle's radius.
// Since area=pi*r*r, we have r = squareroot(area/pi).
public static double getCircleRadius(double area) {
        return Math.sqrt(area/Math.PI);
}
```

```
System.out.println(getCircleRadius(100));
```

Random Class

```
import java.util.Random;

public class RandomExample {

    public static void main(String[] args) {

        Random rand = new Random();
        for (int i = 0; i < 15; i++) {
            System.out.println(rand.nextInt(10));
        }
    }
}
```