# JAVA Basics

Instructor: Vidhya Ramamoorthy
TA:Pavan Kumar Atmuri
Muhammad Saber

# What you will learn

Write your first Java Program

Deconstructing the program

Compiling the program

Variables

Data types

Operators

Type Conversions

Classes

Objects

Variable Scope

Control Structures

Arrays

Classes and Objects

Packages

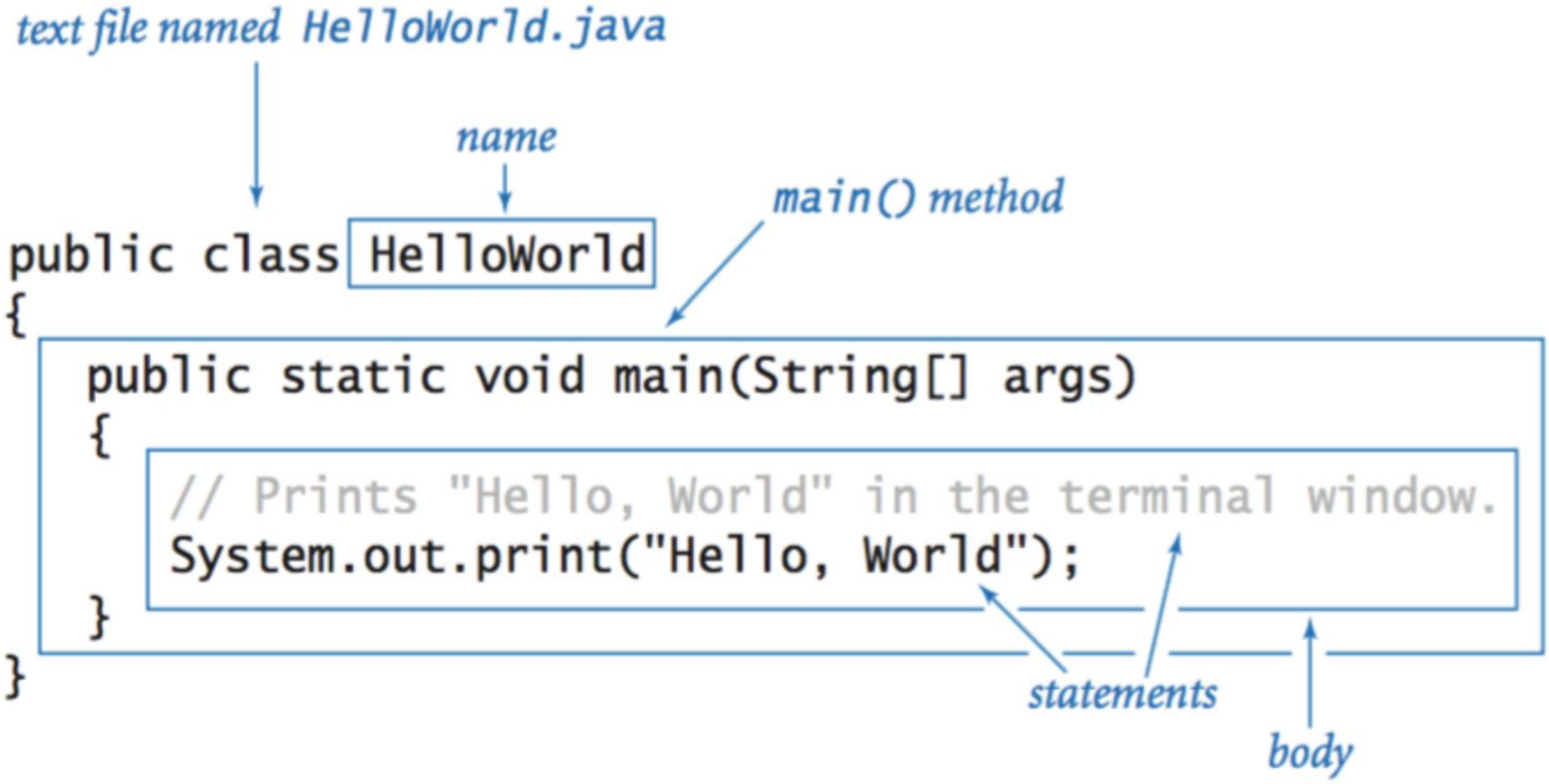
Inheritance

Polymorphism

Abstraction

Interfaces

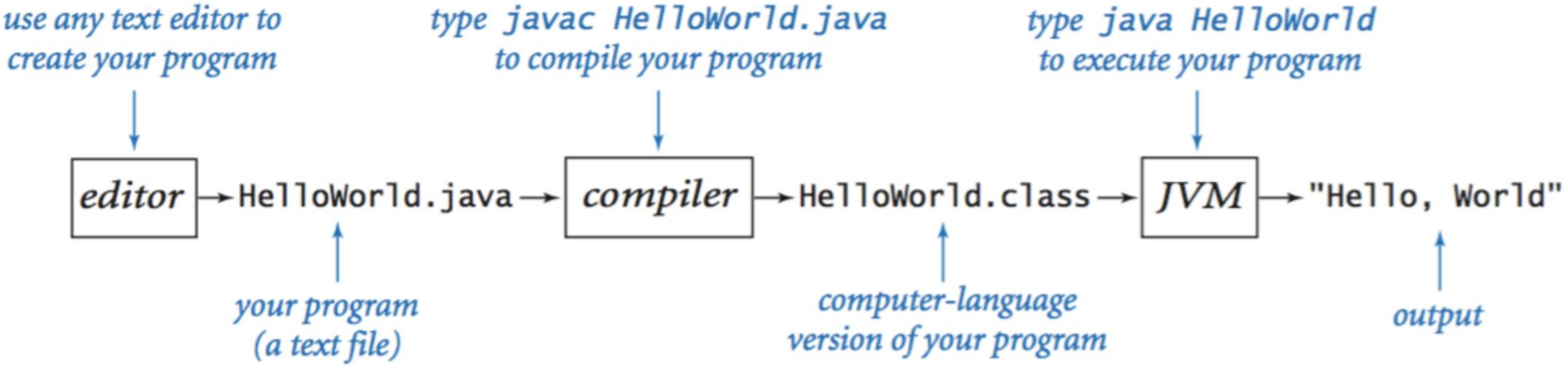
# Hello, World



```
public class HelloWorld
{
   public static void main(String[] args)
   {
      // Prints "Hello, World" in the terminal window.
      System.out.print("Hello, World");
   }
}
```

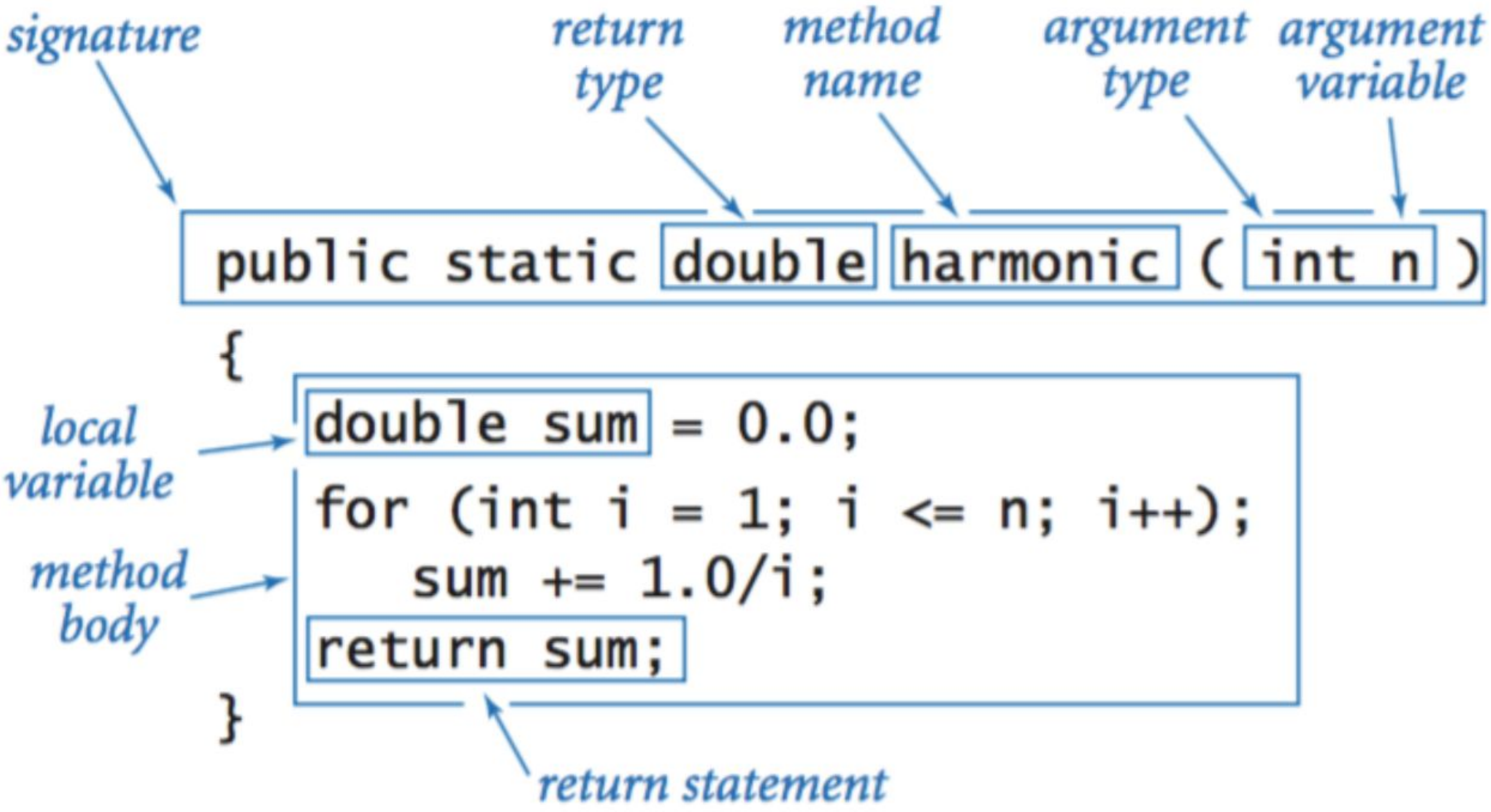
source: https://introcs.cs.princeton.edu/java/lectures

# Editing, Compiling and executing


use any text editor to create your program
type javac HelloWorld.java to compile your program
type java HelloWorld to execute your program
editor
HelloWorld.java
compiler
HelloWorld.class
JVM
"Hello, World"
your program (a text file)
computer-language version of your program
output

source: https://introcs.cs.princeton.edu/java/lectures

# Functions



```
public static double harmonic ( int n )
{
   double sum = 0.0;
   for (int i = 1; i <= n; i++);
      sum += 1.0/i;
   return sum;
}
```

source: https://introcs.cs.princeton.edu/java/lectures

# Variables

- A Storage location with identifier (name),data type & scope.
- The value can change during program execution.
- Name
  - can be Alphanumeric with $ or _ special characters
  - camelcase
  - cannot start with a number
  - cannot have whitespaces

  e.g. int orderNumber;
- Variable must be declared and should be initialized.
  - int orderNumber //variable declaration
  - orderNumber = 1 //variable initialization
  - int orderNumber = 1 //variable declaration & initialization in one statement

# Variable Scope

1. **Member Variables:**variables that are declared inside the class but outside any function and available throughout the class.

**Class Variables:** Member variables defined with static keyword which can be accessed by the class without creating an object of the class.

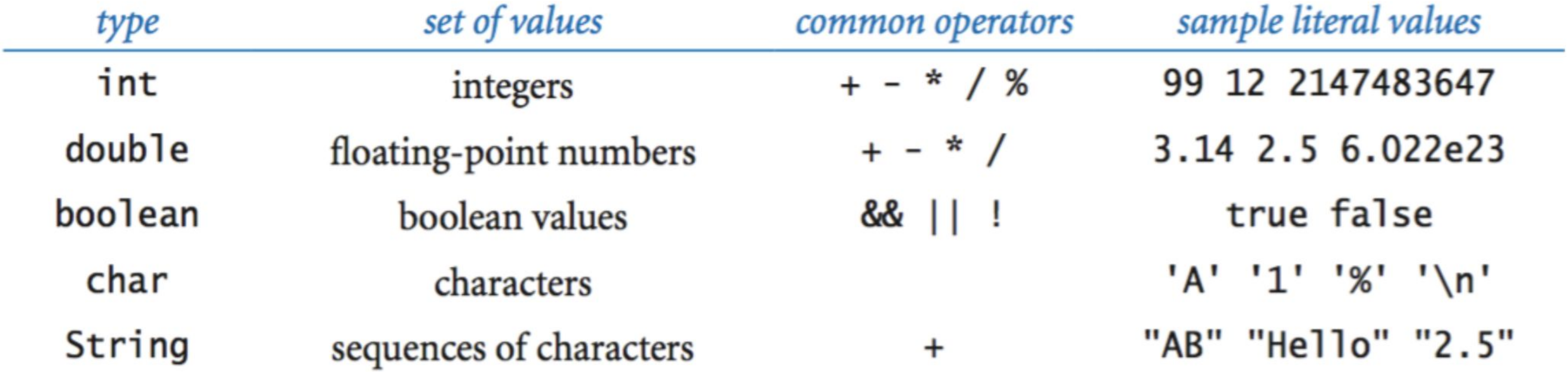
**Instance Variables:**Member variables which can be accessed only by the objects of the class.

2. **Local Variables(Method level scope):** declared inside the method and is accessible only within the method.

```
Public class City{
   static boolean isUSCity = true;
   //Class Variable
   String cityName;
   //Instance Variable
   void currentTimeInCity{
      Date currentTime;
   //Local variable
   }
}
```

# Built-in Data Types

primitive Data types:short, int, long, float, double, boolean, char & byte



| *type* | *set of values* | *common operators* | *sample literal values* |
|---|---|---|---|
| `int` | integers | `+ - * / %` | `99 12 2147483647` |
| `double` | floating-point numbers | `+ - * /` | `3.14 2.5 6.022e23` |
| `boolean` | boolean values | `&& \|\| !` | `true false` |
| `char` | characters | | `'A' '1' '%' '\n'` |
| `String` | sequences of characters | `+` | `"AB" "Hello" "2.5"` |

Commonly used Data types

source: https://introcs.cs.princeton.edu/java/lectures

# Other terminologies

**Literals** : Source code representation of a fixed value (e.g. int pi = 3.14)

**Expression:** Combination of operands and operators whose values need to be computed (e.g. int a = b + c)

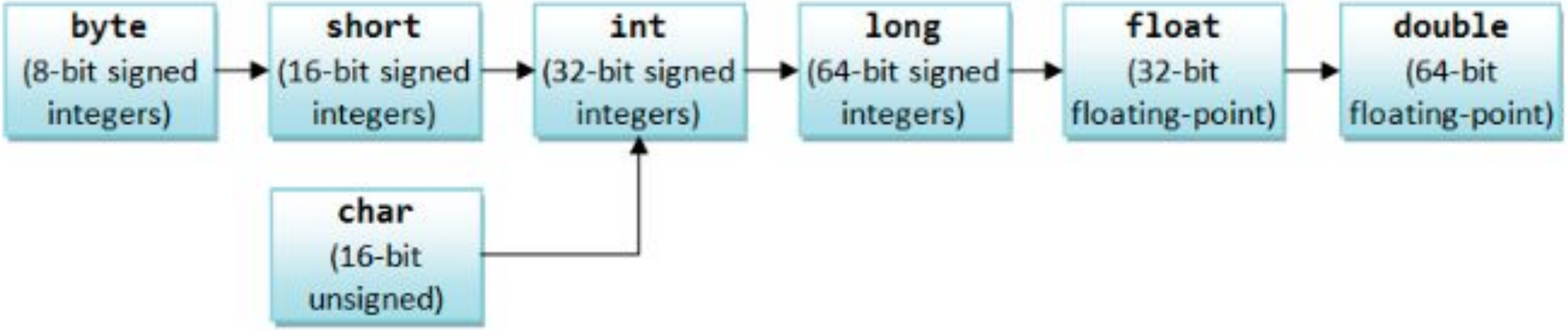
# Type Casting

Operation performed to cast a value from one data type to another.

Explicit Type casting: Explicitly mentioning the data type to cast the value

e.g. i =(int)d to convert a double to integer data type.

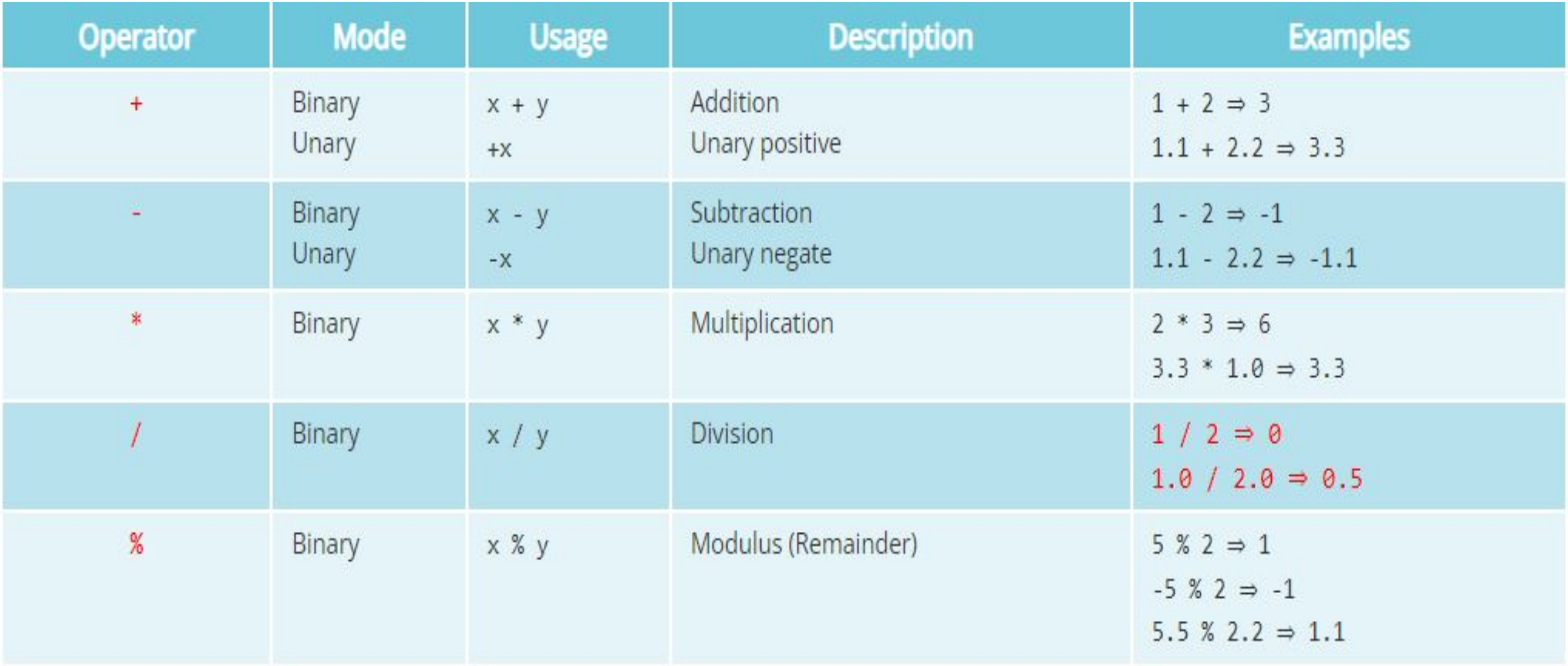
Implicit Type casting: Performed by the compiler automatically if there are no precision loss.



Orders of Implicit Type-Casting for Primitives

https://www3.ntu.edu.sg/home/ehchua/programming/java/j2_basics.html

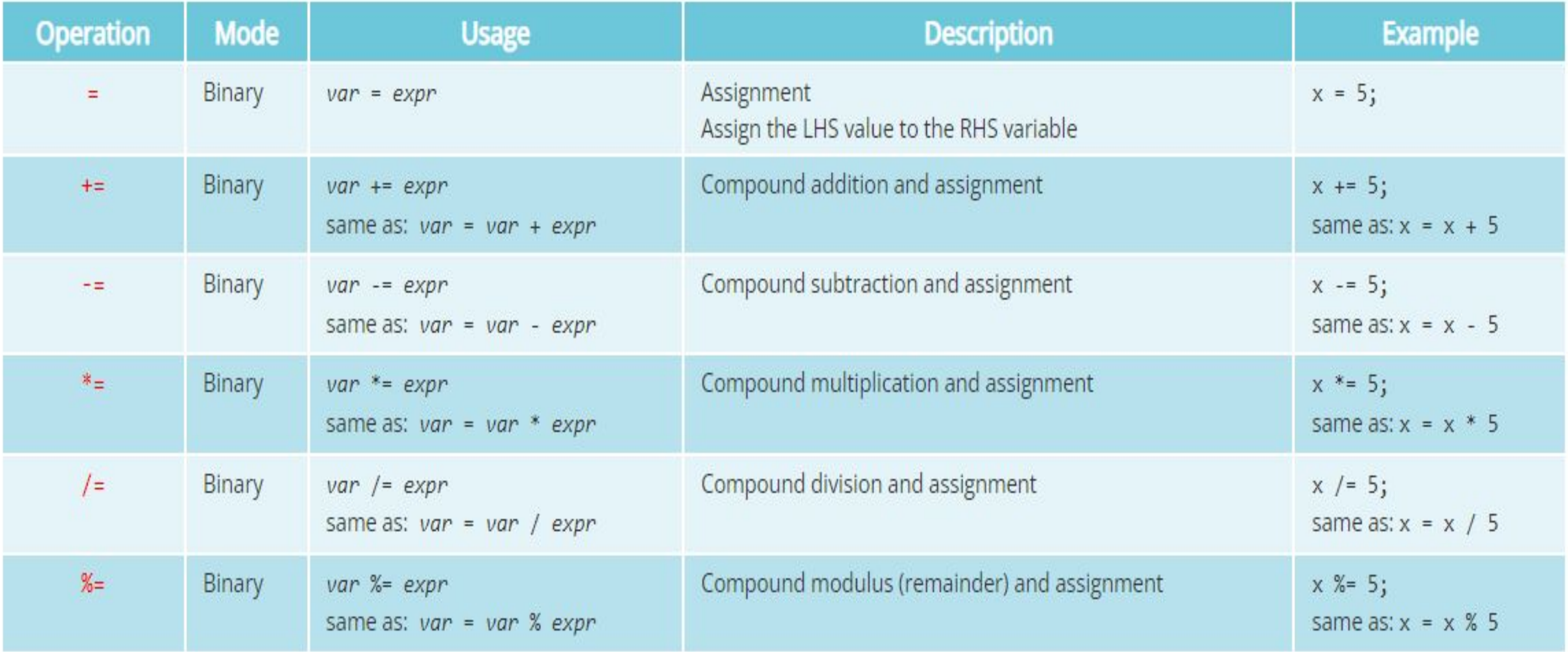
# Operators



| Operator | Mode | Usage | Description | Examples |
|---|---|---|---|---|
| + | Binary<br>Unary | x + y<br>+x | Addition<br>Unary positive | 1 + 2 ⇒ 3<br>1.1 + 2.2 ⇒ 3.3 |
| - | Binary<br>Unary | x - y<br>-x | Subtraction<br>Unary negate | 1 - 2 ⇒ -1<br>1.1 - 2.2 ⇒ -1.1 |
| * | Binary | x * y | Multiplication | 2 * 3 ⇒ 6<br>3.3 * 1.0 ⇒ 3.3 |
| / | Binary | x / y | Division | 1 / 2 ⇒ 0<br>1.0 / 2.0 ⇒ 0.5 |
| % | Binary | x % y | Modulus (Remainder) | 5 % 2 ⇒ 1<br>-5 % 2 ⇒ -1<br>5.5 % 2.2 ⇒ 1.1 |

https://www3.ntu.edu.sg/home/ehchua/programming/java/j2_basics.html

# Operators…



| Operation | Mode | Usage | Description | Example |
|---|---|---|---|---|
| = | Binary | *var* = *expr* | Assignment<br>Assign the LHS value to the RHS variable | x = 5; |
| += | Binary | *var* += *expr*<br>same as: *var* = *var* + *expr* | Compound addition and assignment | x += 5;<br>same as: x = x + 5 |
| -= | Binary | *var* -= *expr*<br>same as: *var* = *var* - *expr* | Compound subtraction and assignment | x -= 5;<br>same as: x = x - 5 |
| *= | Binary | *var* *= *expr*<br>same as: *var* = *var* * *expr* | Compound multiplication and assignment | x *= 5;<br>same as: x = x * 5 |
| /= | Binary | *var* /= *expr*<br>same as: *var* = *var* / *expr* | Compound division and assignment | x /= 5;<br>same as: x = x / 5 |
| %= | Binary | *var* %= *expr*<br>same as: *var* = *var* % *expr* | Compound modulus (remainder) and assignment | x %= 5;<br>same as: x = x % 5 |

https://www3.ntu.edu.sg/home/ehchua/programming/java/j2_basics.html

# Exercise

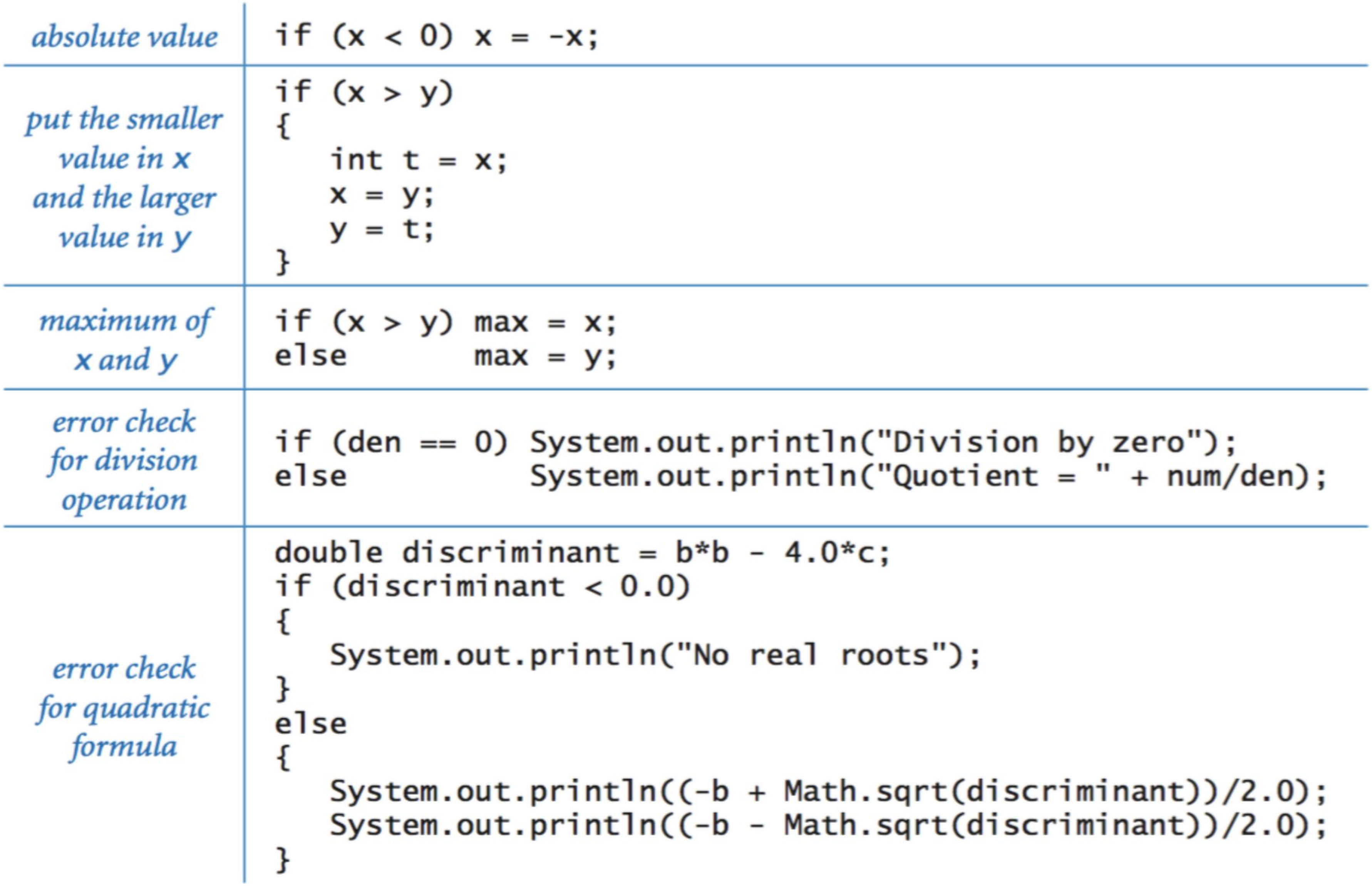
1. Get the radius of the circle from the terminal.
2. Compute the area of a circle with this radius.
3. Print the area in the terminal.

# What you will learn

# Control Structures

1. FOR LOOP
2. WHILE LOOP
3. DO..WHILE LOOP
4. IF
5. SWITCH

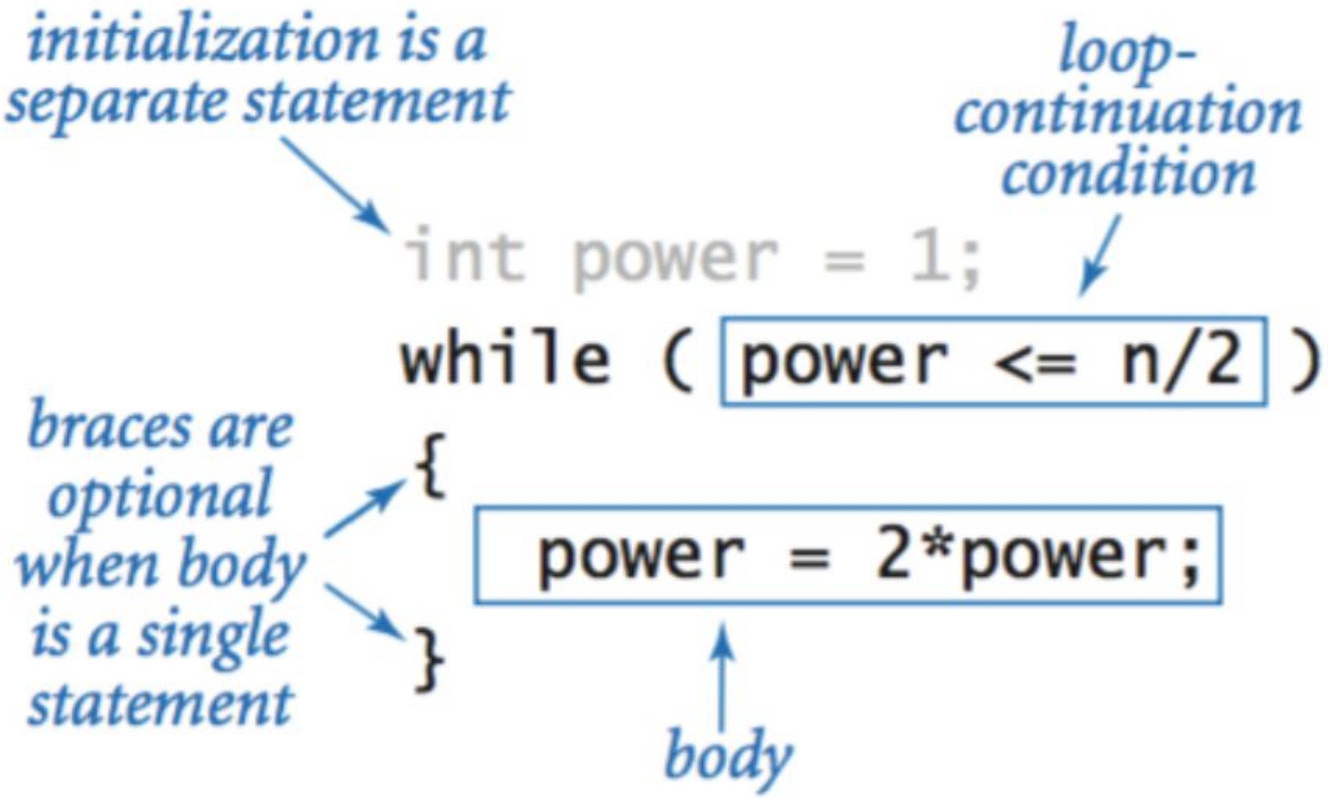
# If and if-else statements.



| | |
|---|---|
| *absolute value* | `if (x < 0) x = -x;` |
| *put the smaller value in x and the larger value in y* | `if (x > y)`<br>`{`<br>`   int t = x;`<br>`   x = y;`<br>`   y = t;`<br>`}` |
| *maximum of x and y* | `if (x > y) max = x;`<br>`else       max = y;` |
| *error check for division operation* | `if (den == 0) System.out.println("Division by zero");`<br>`else          System.out.println("Quotient = " + num/den);` |
| *error check for quadratic formula* | `double discriminant = b*b - 4.0*c;`<br>`if (discriminant < 0.0)`<br>`{`<br>`   System.out.println("No real roots");`<br>`}`<br>`else`<br>`{`<br>`   System.out.println((-b + Math.sqrt(discriminant))/2.0);`<br>`   System.out.println((-b - Math.sqrt(discriminant))/2.0);`<br>`}` |

source: https://introcs.cs.princeton.edu/java/lectures

# Anatomy of a while loop



```
int power = 1;
while ( power <= n/2 )
{
   power = 2*power;
}
```

initialization is a separate statement

loop-continuation condition

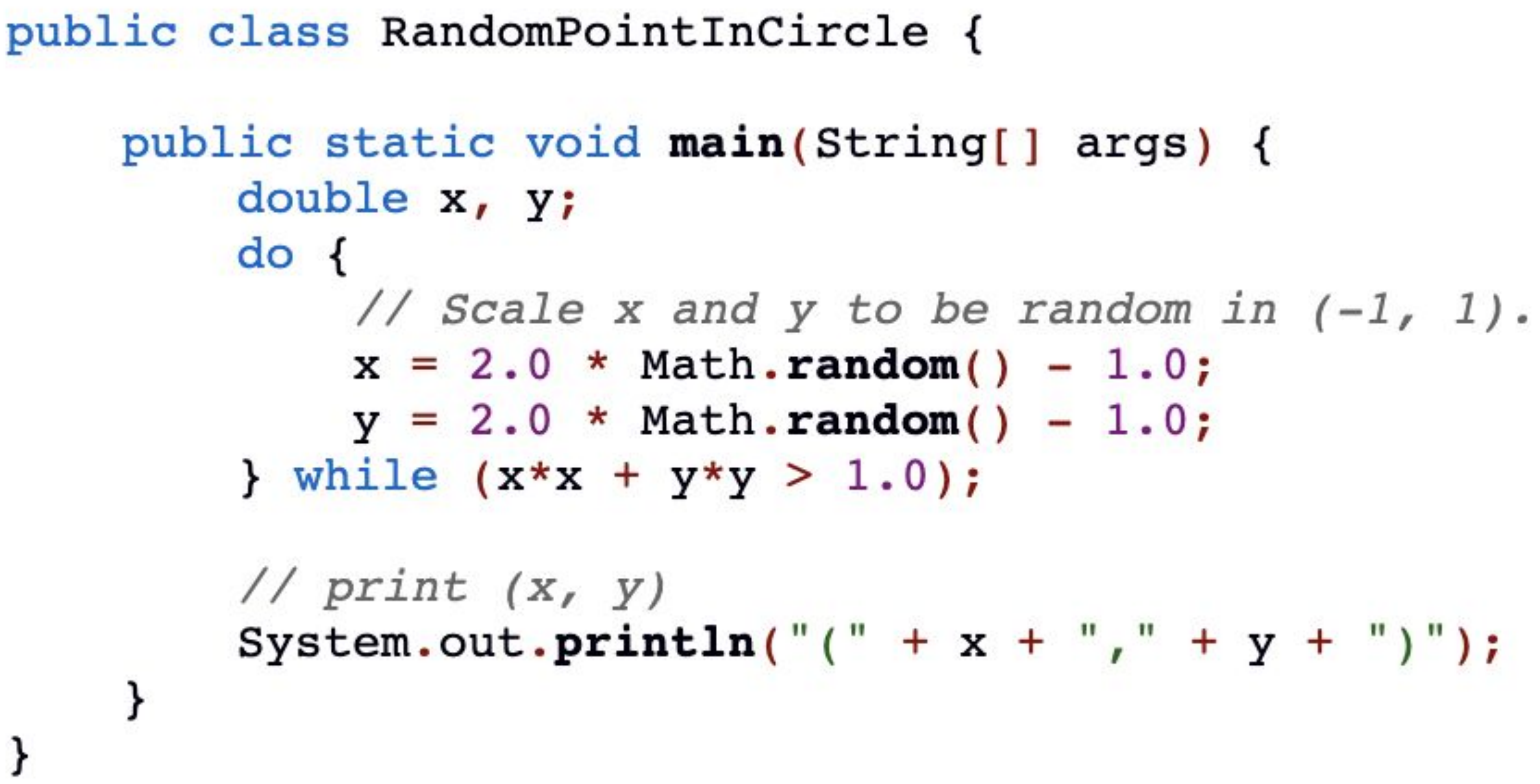
braces are optional when body is a single statement

body

source: https://introcs.cs.princeton.edu/java/lectures

# Do While Loop

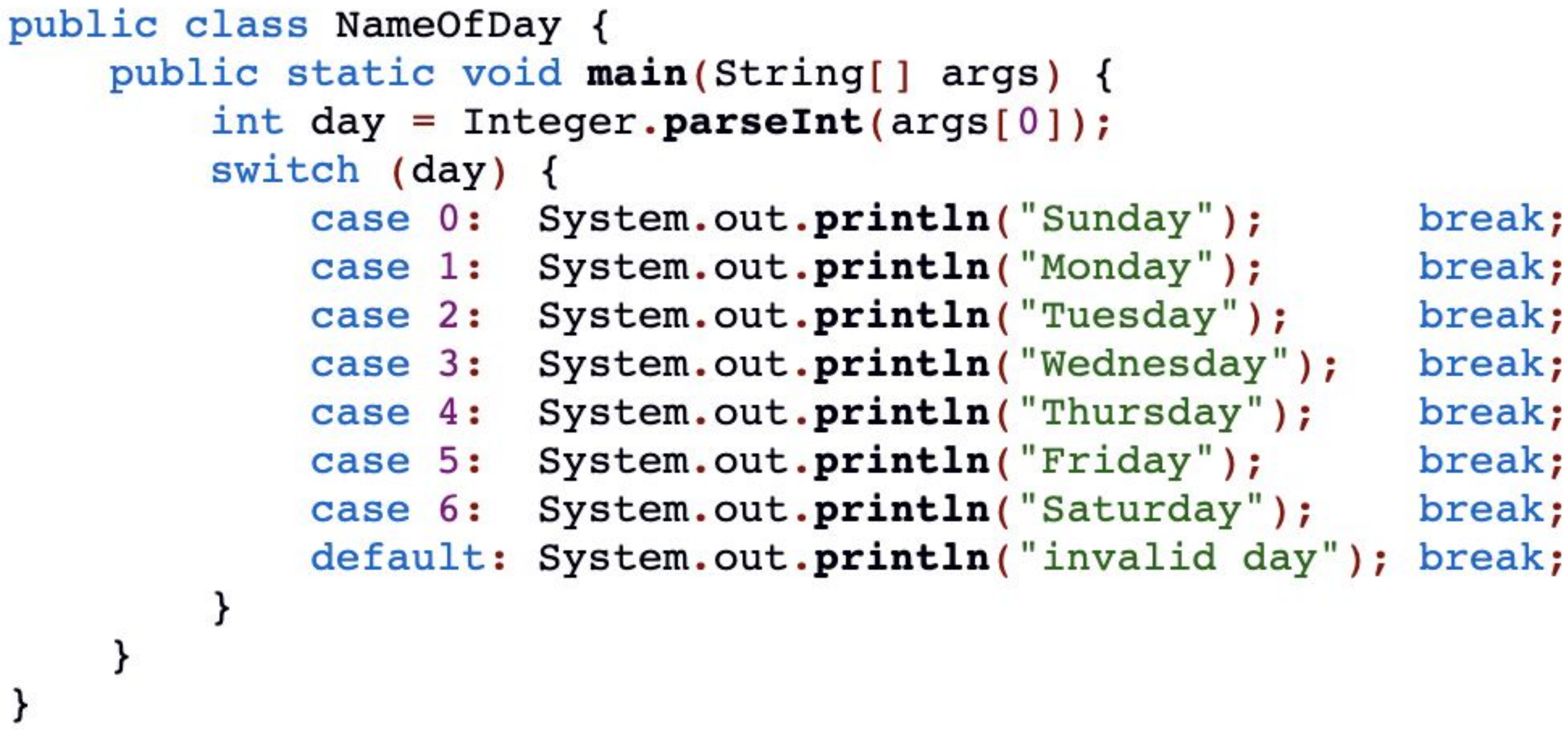
Loop condition checked only after first execution of the loop



```java
public class RandomPointInCircle {

    public static void main(String[] args) {
        double x, y;
        do {
            // Scale x and y to be random in (-1, 1).
            x = 2.0 * Math.random() - 1.0;
            y = 2.0 * Math.random() - 1.0;
        } while (x*x + y*y > 1.0);

        // print (x, y)
        System.out.println("(" + x + "," + y + ")");
    }
}
```

https://introcs.cs.princeton.edu/java/13flow/RandomPointInCircle.java.html

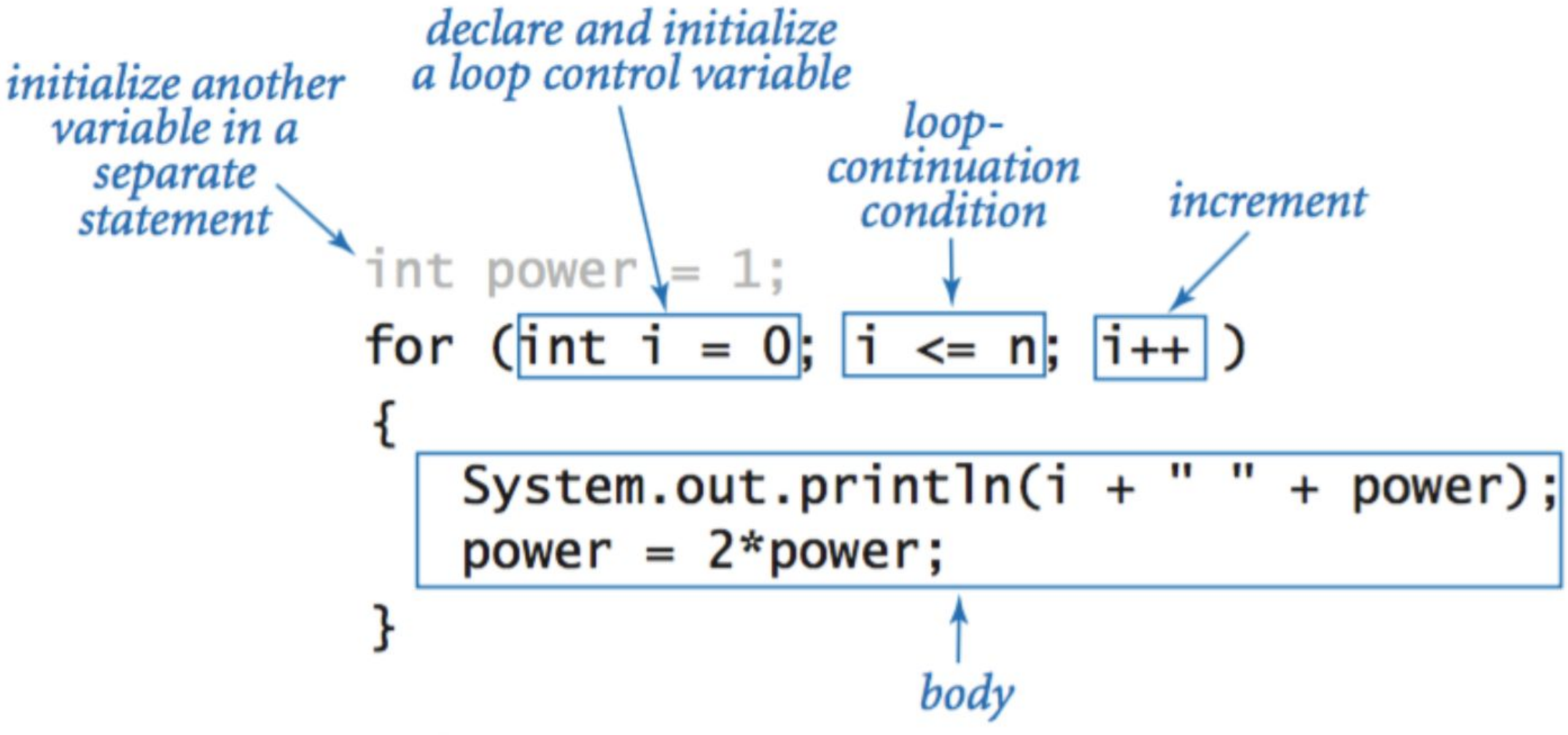
# SWITCH



```java
public class NameOfDay {
    public static void main(String[] args) {
        int day = Integer.parseInt(args[0]);
        switch (day) {
            case 0:  System.out.println("Sunday");      break;
            case 1:  System.out.println("Monday");      break;
            case 2:  System.out.println("Tuesday");     break;
            case 3:  System.out.println("Wednesday");   break;
            case 4:  System.out.println("Thursday");    break;
            case 5:  System.out.println("Friday");      break;
            case 6:  System.out.println("Saturday");    break;
            default: System.out.println("invalid day"); break;
        }
    }
}
```

https://introcs.cs.princeton.edu/java/13flow/NameOfDay.java.html

# Anatomy of a For Loop



```
int power = 1;
for (int i = 0; i <= n; i++ )
{
   System.out.println(i + " " + power);
   power = 2*power;
}
```

initialize another variable in a separate statement
declare and initialize a loop control variable
loop-continuation condition
increment
body

source: https://introcs.cs.princeton.edu/java/lectures

# Java Features

(Character Counting application)

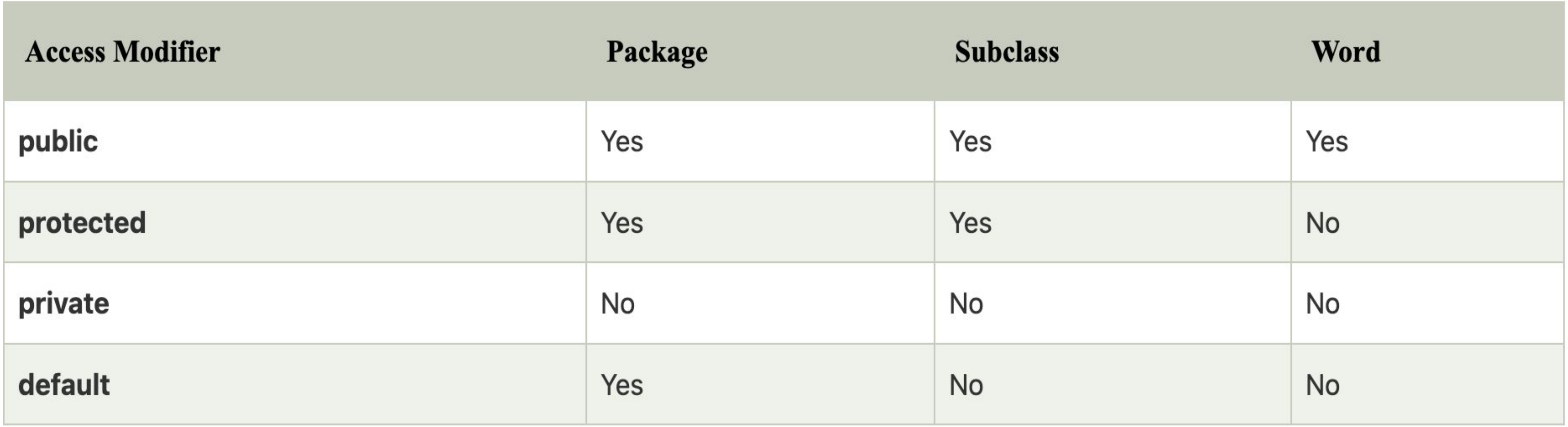
```
class Count {
    public static void main(String[] args)
        throws java.io.IOException
    {
        int count = 0;
        while (System.in.read() != -1)
            count++;
        System.out.println("Input has " + count + " chars.");
    }
}
```

# Operators

- int count **=** 0;
- System.in.read() **!=** -1
- count **++**;
- "Input has " **+** count **+** " chars."

# Access Modifiers

Access Modifiers determine the visibility of a field and method defined in a class to other classes.



| Access Modifier | Package | Subclass | Word |
|---|---|---|---|
| **public** | Yes | Yes | Yes |
| **protected** | Yes | Yes | No |
| **private** | No | No | No |
| **default** | Yes | No | No |

source: https://www.javatpoint.com/scope-of-variables-in-java

# Object Oriented Programming - Why?

Modularity: Implementation of each object can be done independently.

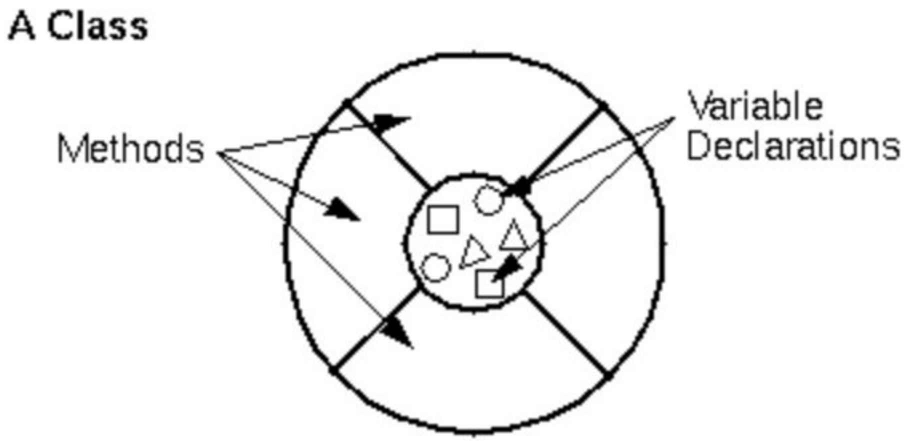
Information Hiding: Exposes only what is required via methods and hides other internal details.

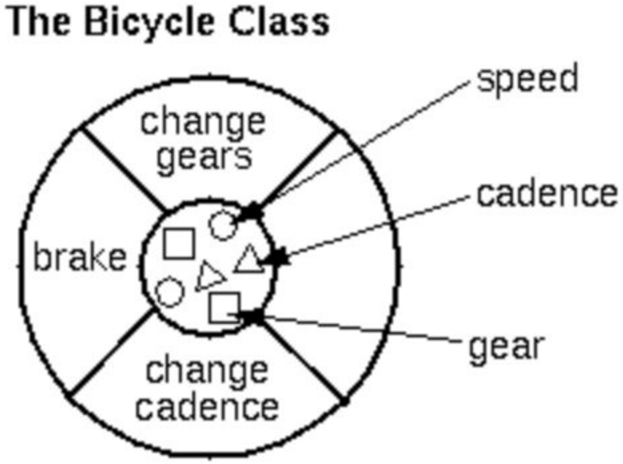
Reusability: Reuse objects if already exists.

Easy debugging: Enables developers to remove an object with issue and include another object.

# Classes

- A class is a blueprint or prototype that defines the variables and methods common to all objects of a certain kind.





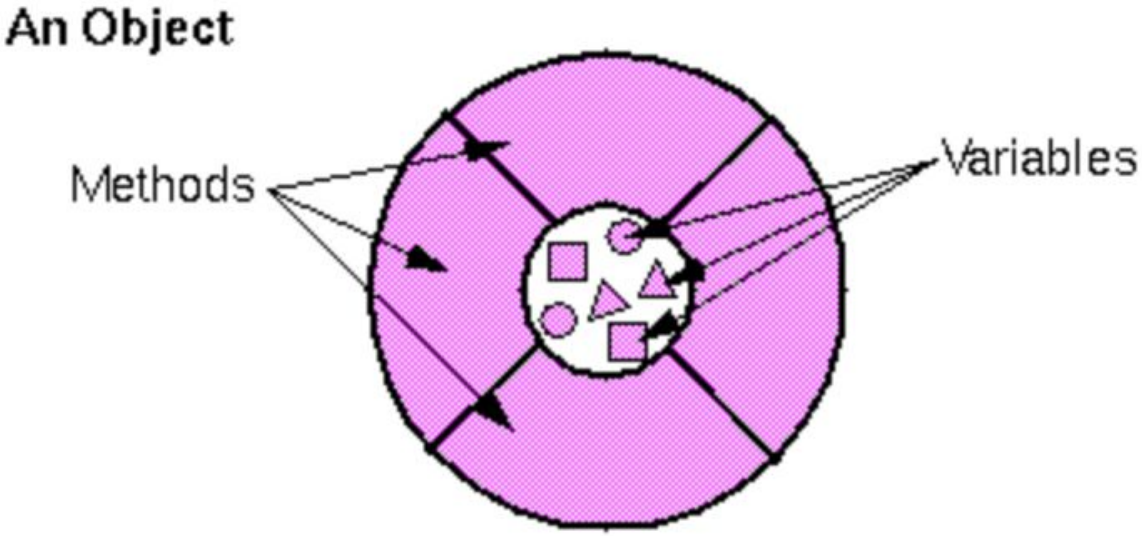
https://docs.oracle.com/javase/tutorial/java/concepts/class.html

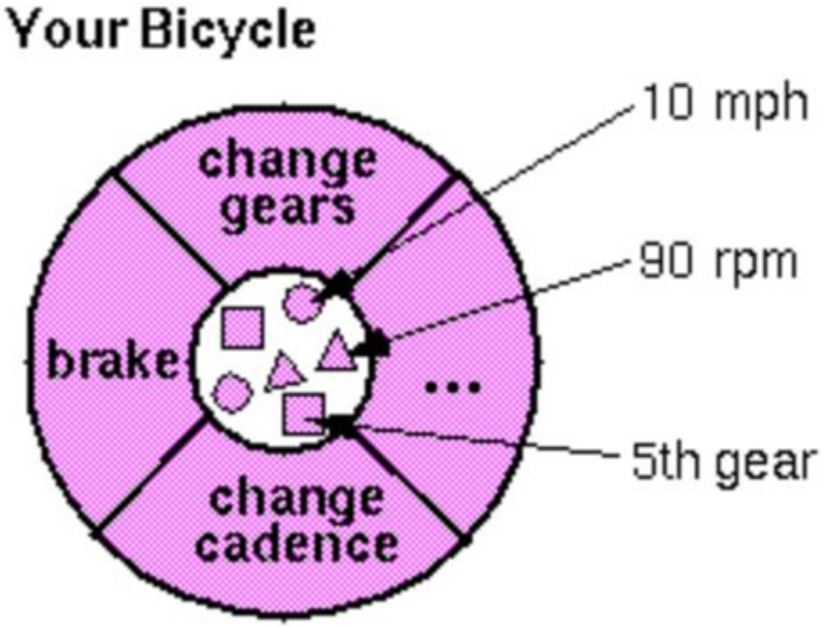
# Object

## Characteristics:

1) State
2) Behavior

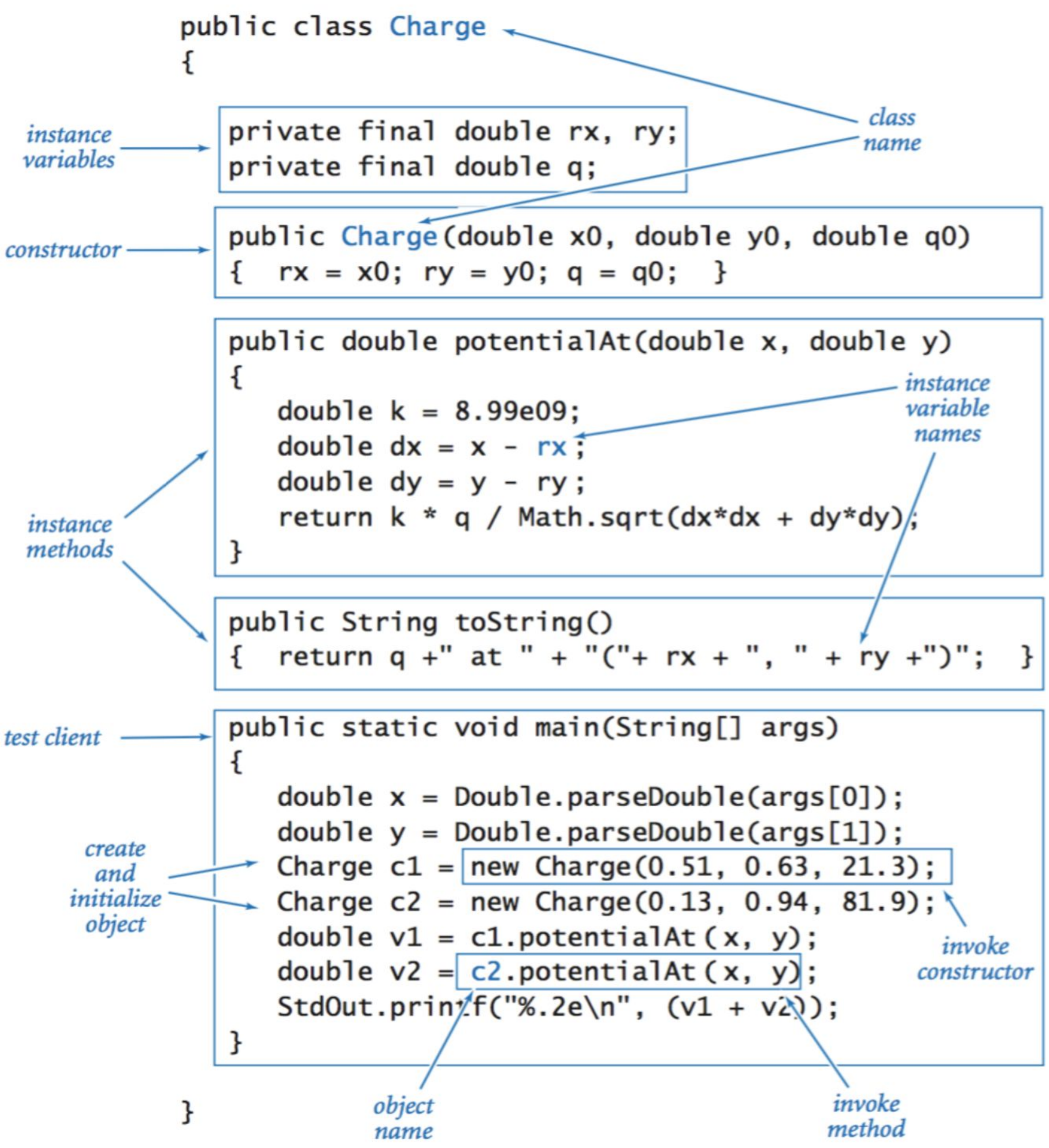
e.g:
Object :Bicycle
State :current gear, current pedal cadence, two wheels, number of gears
Behavior :braking, accelerating, slowing down, changing gears





https://docs.oracle.com/javase/tutorial/java/concepts/class.html

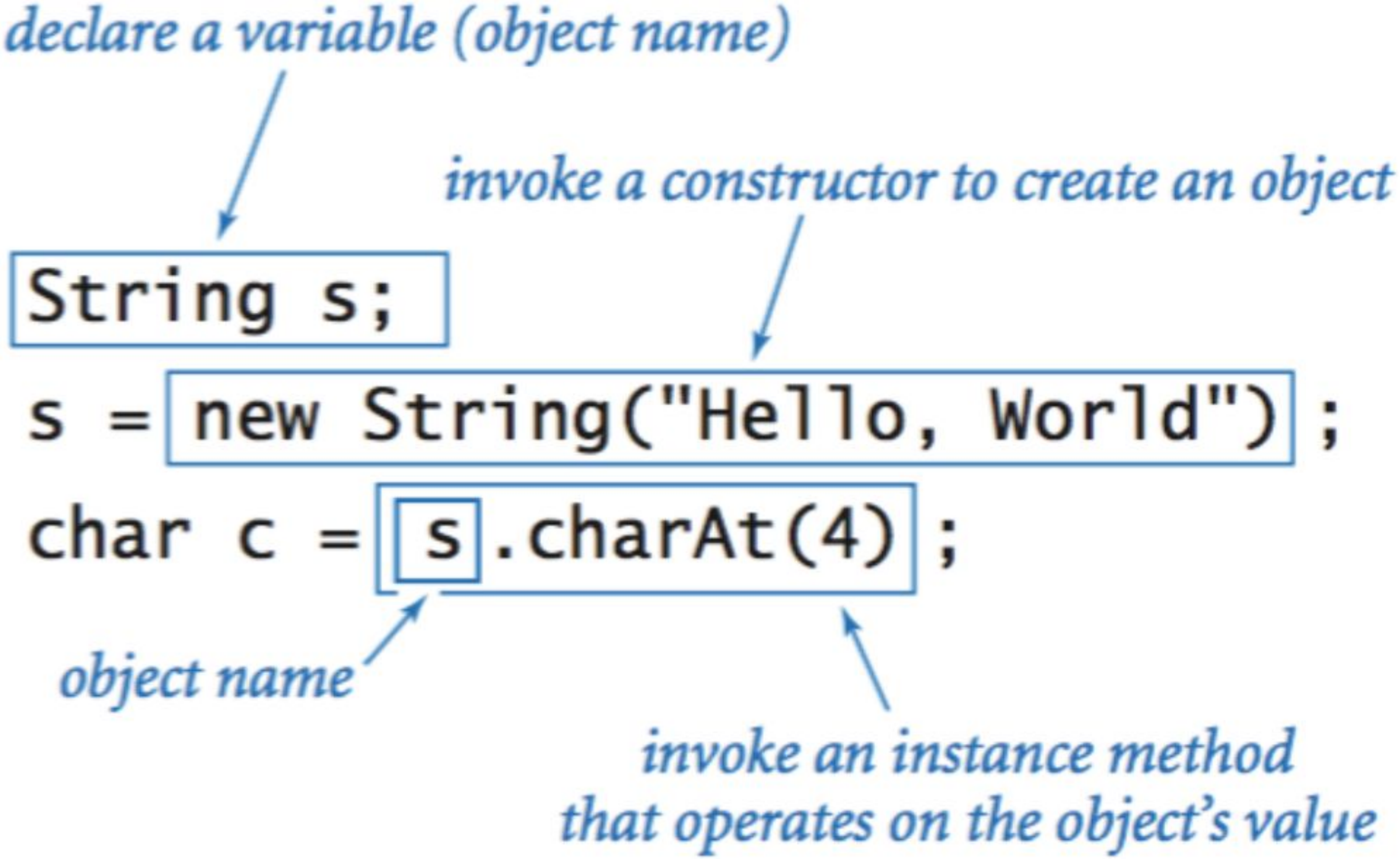
# Sample class



```
public class Charge
{
   private final double rx, ry;
   private final double q;

   public Charge(double x0, double y0, double q0)
   {  rx = x0; ry = y0; q = q0;  }

   public double potentialAt(double x, double y)
   {
      double k = 8.99e09;
      double dx = x - rx;
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx + dy*dy);
   }

   public String toString()
   {  return q +" at " + "("+ rx + ", " + ry +")";  }

   public static void main(String[] args)
   {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(0.51, 0.63, 21.3);
      Charge c2 = new Charge(0.13, 0.94, 81.9);
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);
      StdOut.printf("%.2e\n", (v1 + v2));
   }

}
```

source: https://introcs.cs.princeton.edu/java/lectures

# Using an object.



*declare a variable (object name)*

*invoke a constructor to create an object*

```
String s;
s = new String("Hello, World") ;
char c = s.charAt(4) ;
```

*object name*

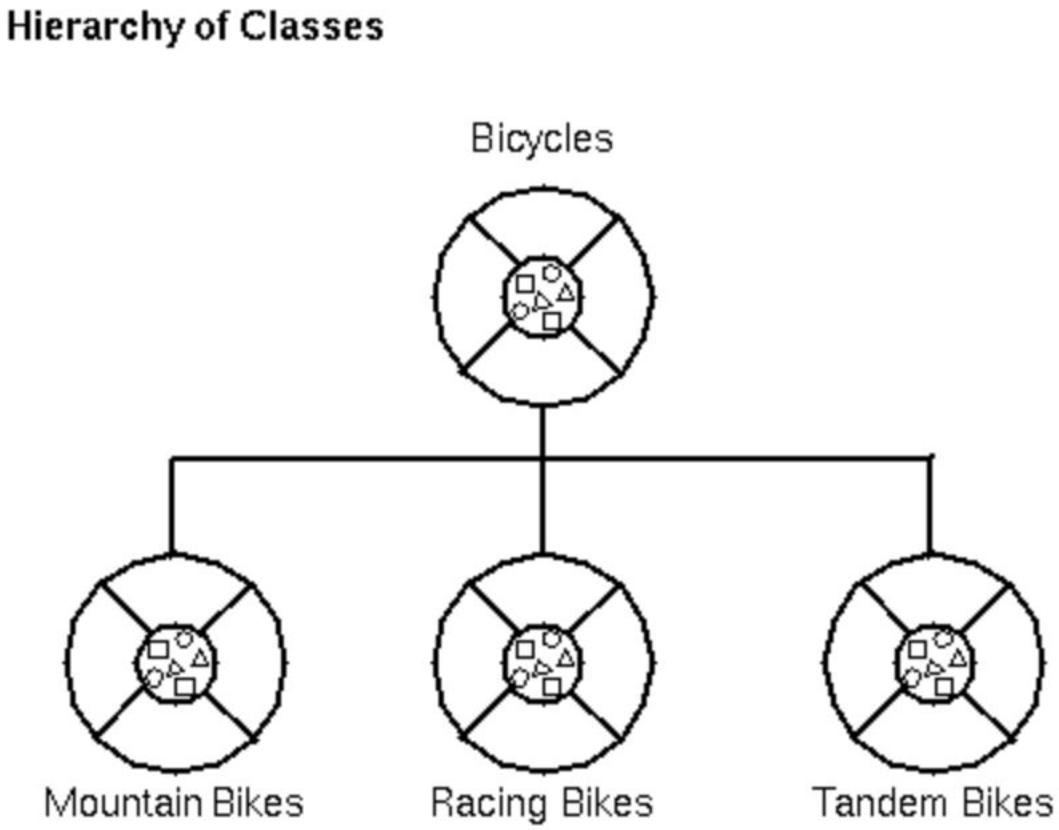
*invoke an instance method*
*that operates on the object's value*

source: https://introcs.cs.princeton.edu/java/lectures

# Inheritance

- A subclass inherits methods and variables from superclass
- implemented using the keyword **extends**
- java.lang.Object class is the base class of all classes in Java

class MountainBikes **extends** Bicycles{
….}



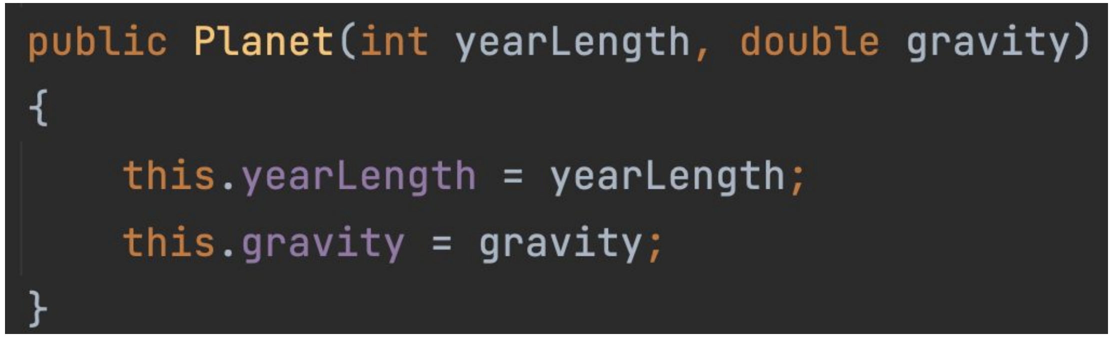
https://docs.oracle.com/javase/tutorial/java/concepts/class.html

# Encapsulation

- ability of an object to hide its data and methods from outside world.
- an object's variables are packaged within the protective custody of its methods
- achieved via access modifiers public, private, protected.
- Benefits
  - Modularity
  - Information Hiding

# Constructors

- Special methods in Classes used in Object Creation
- Similar to other method declaration
- Use name of the class as the method name
- Do not have return type
- A class can have multiple constructors with different argument and different return type.
- if a constructor is not explicitly specified then a default constructor with no parameter is automatically added to the class.



```
public Planet(int yearLength, double gravity)
{
    this.yearLength = yearLength;
    this.gravity = gravity;
}
```

# Polymorphism

Ability of a class to provide different implementations of a method depending on the type of object that is passed to the method.

## Types:

1. **Method Overloading:**
   - Multiple uniquely working methods created with the same name in the same class.
   - Variations in return type, number, type and order of parameters.
   - Happens at compile time.
2. **Method Overriding:**
   - Subclass or a child class has the same method as declared in the parent class.
   - Happens at run time based on the object which is referred by the reference variable.

# Interfaces

- Contract between the class and the outside world.
- Group of related methods represented with empty body statements.
- contain method signatures, default methods, static methods and constant definitions.
- All methods in interface needs to be implemented by the class using the interface.
- Compilation Error will occur if the methods are not implemented.
- Interfaces can’t be instantiated.
- can only be implemented and extended.
- can be used anywhere a data type can be used.

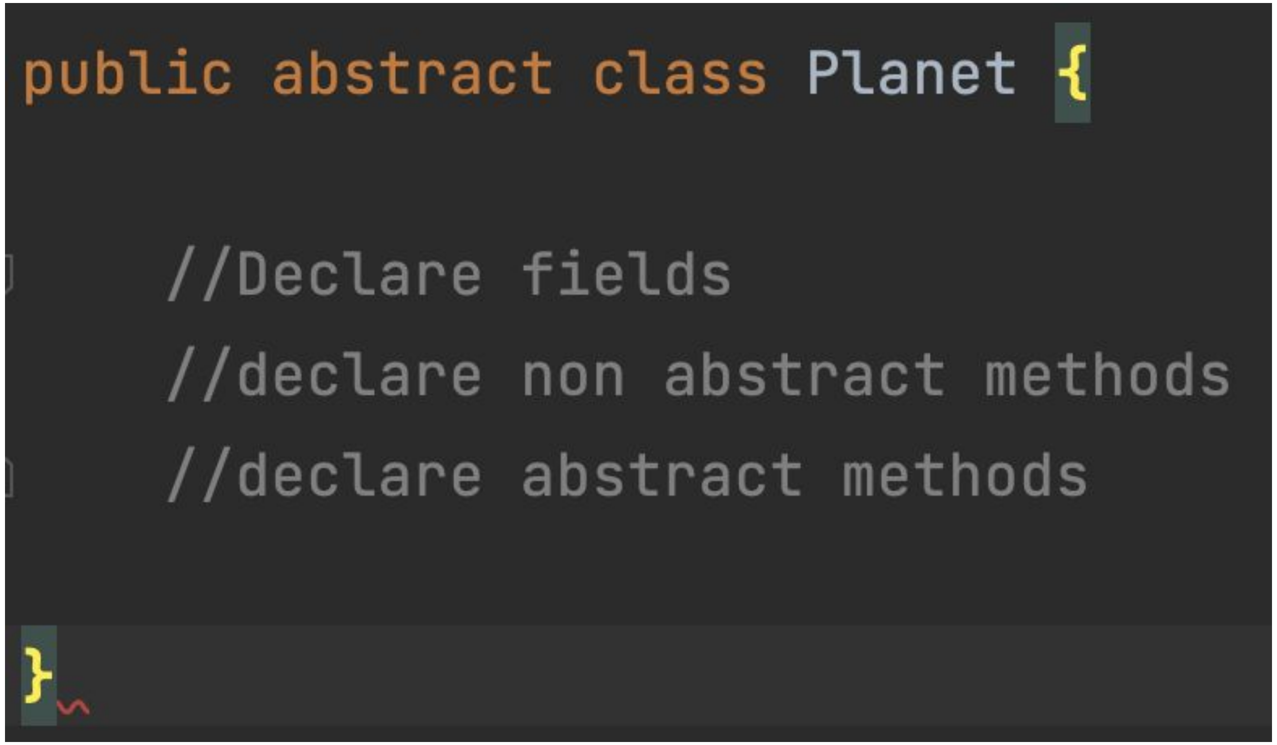
```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

```
class GiantBicyle implements Bicycle {
}
```

https://docs.oracle.com/javase/tutorial/java/concepts/class.html

# Abstract Classes

- Classes declared with the keyword abstract.
- Cannot be instantiated.
- can be subclassed.



```
public abstract class Planet {

    //Declare fields
    //declare non abstract methods
    //declare abstract methods

}
```

# Interfaces vs Abstract classes

- Use Abstract classes when:
  - Code is shared among related classes.
  - Classes extending abstract class will have common methods, fields and requires access modifiers like protected and private.
  - Need to have methods that modifies the object state via fields that are not static or final.
- Use Interfaces when:
  - you expect unrelated classes to share common method signatures.
  - you don’t care about who implements the behavior but only want to specify the behavior.
  - have a need for multiple inheritance which is not achieved through class hierarchy.

# Abstraction

Process of separating ideas or behavior of a class from specific implementation details.

achieved via interfaces and Abstract classes.

Encapsulation focuses on implementation of object’s behavior.

Abstraction focuses on the behavior itself.

# Classes - Types

- Abstract - classes that can’t be instantiated but inherited and declared with the keywork ‘abstract’.
- Final Class - classes declared final can’t be subclasses and are immutable.
- Concrete Class - normal classes with implementation for all the methods
- Singleton Class - A class with only one object.
- POJO Class - Plain Old Java Object class which has only private variables with setters and getters.
- Inner Class
  - Anonymous - inner class with no name.
  - Local - inner class defined within a block in a class.
  - Static - nested classes that provide blueprint for inherited class and can have only static members. Objects can’t be created for a static class.

# Anonymous classes

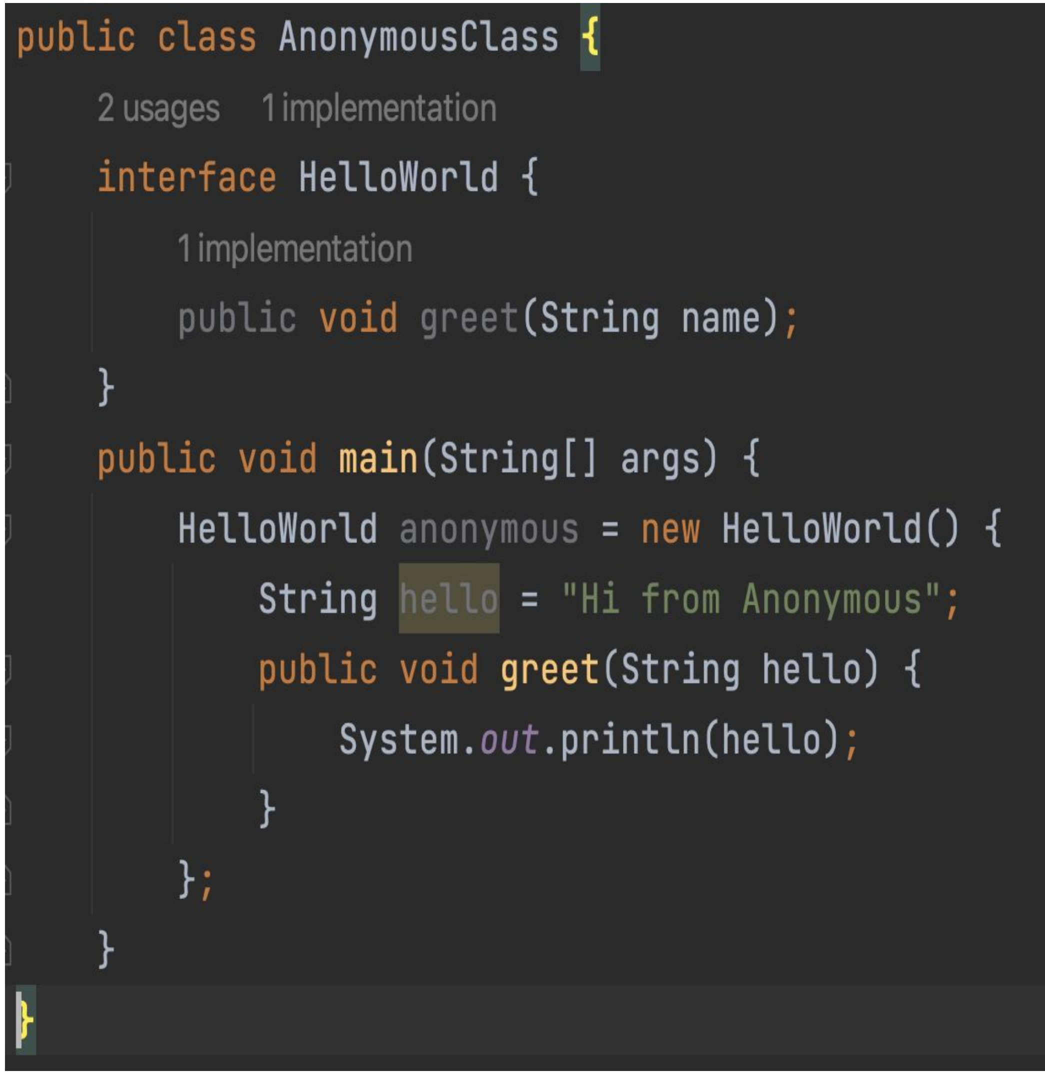
- Inner class within a class created without a name.
- Only one object is created for an anonymous class.

Syntax:

- new Operator
- name of the interface
- parentheses that contain constructor argument
- body

Advantages:

- Enables class declaration and instantiation at the same time.



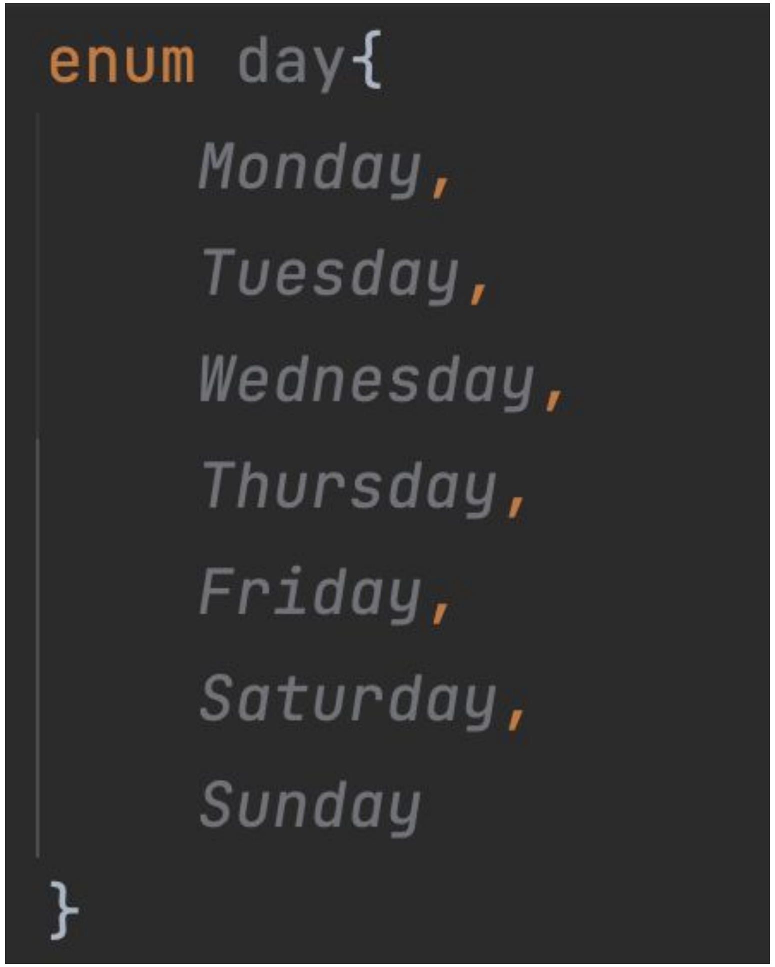
```
public class AnonymousClass {
    2 usages   1 implementation
    interface HelloWorld {
        1 implementation
        public void greet(String name);
    }
    public void main(String[] args) {
        HelloWorld anonymous = new HelloWorld() {
            String hello = "Hi from Anonymous";
            public void greet(String hello) {
                System.out.println(hello);
            }
        };
    }
}
```

# Static classes

- Only nested classes can be declared static.
- defined with the keyword static.
- class can’t be instantiated.
- can access static members of the outer class.
- can access non static members of the outer class only via object reference.

# Enumerations

- A data type which represents a group of constants.
- Defined using the keyword enum.
- A variable declared with enum type also represents enum class.
- Enum class can have its own constructor, properties and methods.
- extends java.lang.Enum class and hence can’t inherit any other class.



```
enum day{
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday,
        Sunday
}
```

# Strings

- Strings are objects not primitive types.
- String class is used to create and manipulate strings.
- String class is immutable.
- String literals
  - String = “123” creates string literal.
  - stored in a special memory pool in heap memory called String pool.
- String Objects
  - String s = new String(“123”) creates String Objects.
  - stored in heap memory
- Interning - the process of storing only one copy of a string literal.

# Null vs Empty Strings

- null string is not the same empty string
- null string -
  - represented by null.
  - has no value at all.
- empty string -
  - represented as ‘’.
  - character sequence of zero characters.

# String methods

- length() - returns the length of the string including white spaces.
- concat() -used for concatenating two strings.
- equals() - used for comparing strings. // equality operator won’t work with string.
- substring() - get
- split() - used for splitting string using regular expression.
- replace() /replaceall() - replace characters in String.
- matches() - search for a regular expression pattern inside string.
- charAt()-returns the char at given index

# Commenting

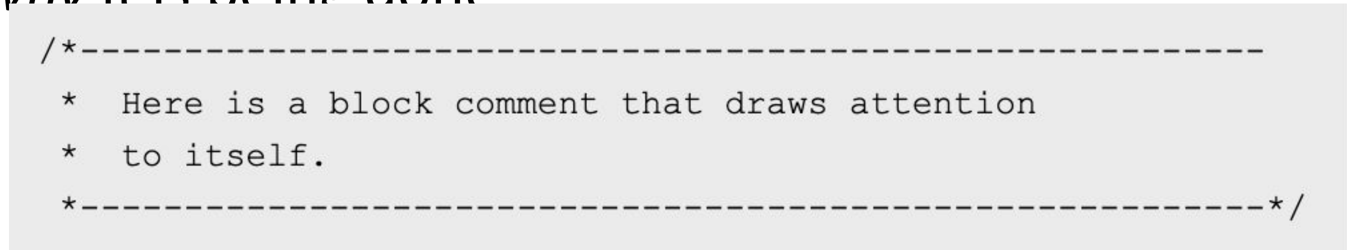
## Good programmers write code that documents itself.

As a general rule,

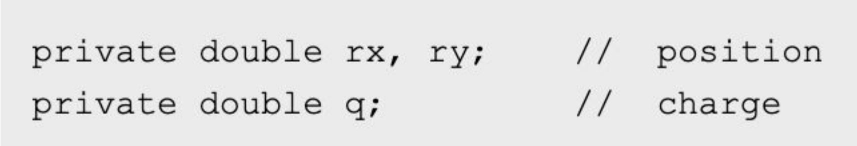
the code explains to the computer and programmer *what* is being done;

the comments explain to the programmer *why* it is being done

- Block Comments
- Line Comments
- Bold Comments (e.g.JavaDoc)

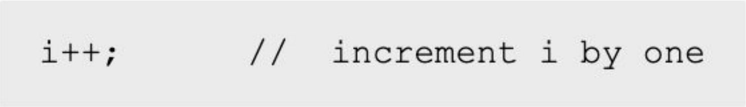


```
/*--------------------------------------------------------------
 *  Here is a block comment that draws attention
 *  to itself.
 *------------------------------------------------------------*/
```

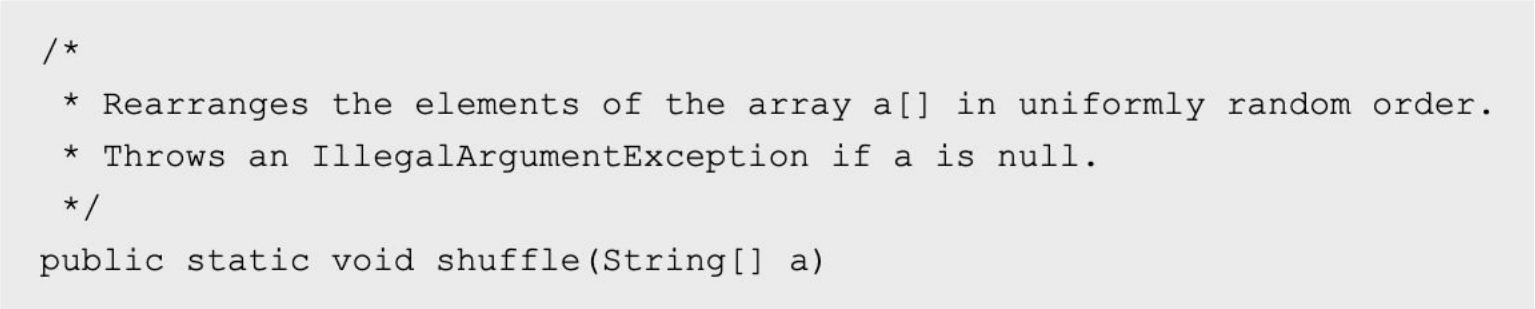


```
private double rx, ry;    //  position
private double q;         //  charge
```

Don't do this



```
i++;      //  increment i by one
```



```
/*
 * Rearranges the elements of the array a[] in uniformly random order.
 * Throws an IllegalArgumentException if a is null.
 */
public static void shuffle(String[] a)
```

# Packages

- Packages or Namespaces are nothing but directories.
- Classes serving some common purpose are grouped together under one package.
- Every class belongs to a package.
- If you want to access classes from other packages, then you need to import them.

  e.g. *java.util* is a package. the classes in this package are accessed by

*import java.util.**;

- All classes can access other classes in the same package.
- All classes can access classes in java.lang package. e.g. java.lang.System

# JDK vs JRE vs JVM

## JVM - Java Virtual Machine

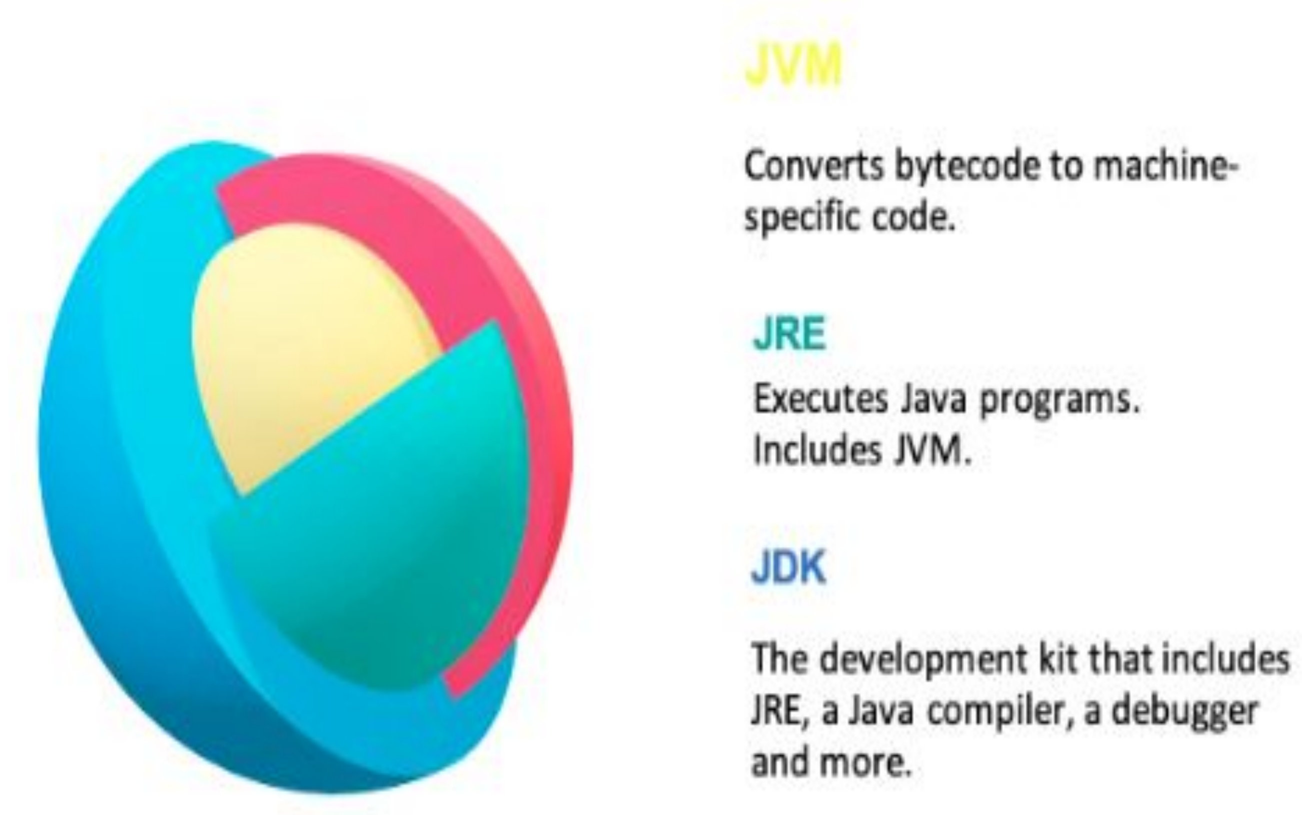
- loads, verifies and executes Java bytecode.
- also known as Interpretor.
- responsible for converting bytecode to machine-specific code.

## JRE - Java Runtime Environment

- set of software tools responsible for execution of the Java program or application on your system.
- required by the end user to run any Java application.

## JDK - Java Development Kit

- component that enables Java development.
- contains all the tools required to compile, debug and run a program developed using the Java platform



JVM

Converts bytecode to machine-specific code.

JRE

Executes Java programs. Includes JVM.

JDK

The development kit that includes JRE, a Java compiler, a debugger and more.

source: https://www.ibm.com/cloud/blog/jvm-vs-jre-vs-jdk

# JDK vs JRE vs JVM Differences

- JDK is the development platform, while JRE is for execution.
- JVM is the foundation, or the heart of Java programming language, and ensures the program’s Java source code will be platform-agnostic.
- JVM is included in both JDK and JRE – Java programs won’t run without it.

source: https://www.ibm.com/cloud/blog/jvm-vs-jre-vs-jdk

# Writing Clear Code

- Coding
- Naming Convention
- Comments
- Whitespace
- Indenting

# Coding

- Keep programs and methods short and manageable.
- Use language-specific idioms.
- Use straightforward logic and flow-of-control.
- Avoid magic numbers (numbers other than –1, 0, 1, and 2); instead, give them meaningful symbolic names.

# Naming Convention

| IDENTIFIER | NAMING RULES | EXAMPLE |
|---|---|---|
| Variables | A short, but meaningful, name that communicates to the casual observer what the variable represents, rather than how it is used. Begin with a lowercase letter and use camel case (mixed case, starting with lower case). | mass<br>hourlyWage<br>isPrime |
| Constant | Use all capital letters, separating internal words with the underscore character. | BOLTZMANN<br>MAX_HEIGHT |
| Class | A noun that communicates what the class represents. Begin with an uppercase letter and use camel case for internal words. | Complex<br>Charge<br>PhoneNumber |
| Method | A verb that communicates what the method does. Begin with a lowercase letter and use camelCase for internal words. | move()<br>draw()<br>enqueue() |

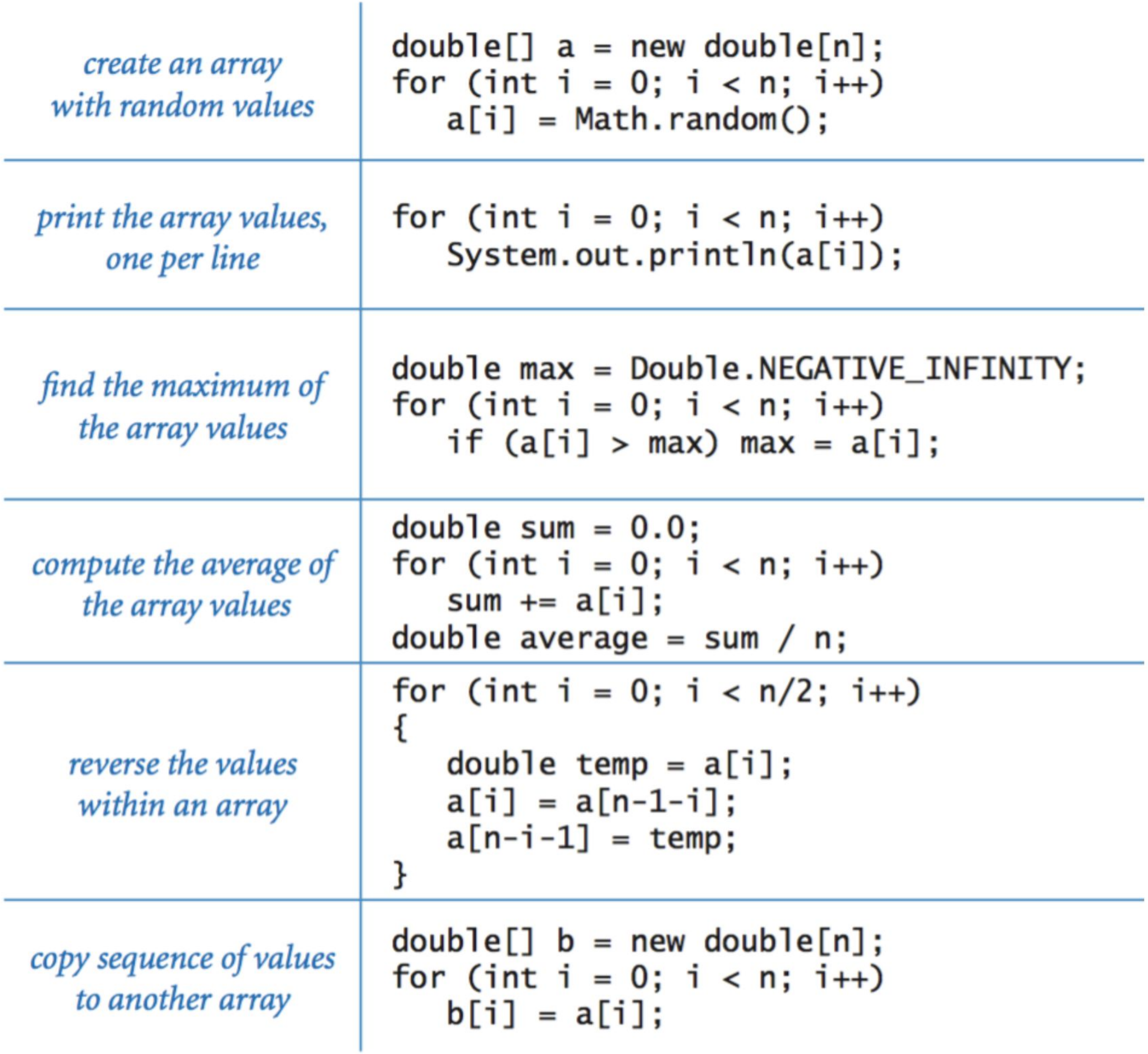
# Defining Classes

- Declare the fields and methods as private.
- Expose only the methods if you have a need.
- if you need to access a field in a class, create a method to pass the variable when the method is called.

# Errors

- Compile Time: Errors thrown by the compiler as the program can’t run without correcting these errors. e.g missing a semicolon.
- Run Time:Errors detected during the execution of the program.These occur when the program is expecting the computer to do something which the computer is unable to perform. e.g. typecasting between incompatible data types.
- Logical:These are semantic errors which is caused due to incorrect logic implemented in the program. e.g.checking for whether 1 is equal to 2.

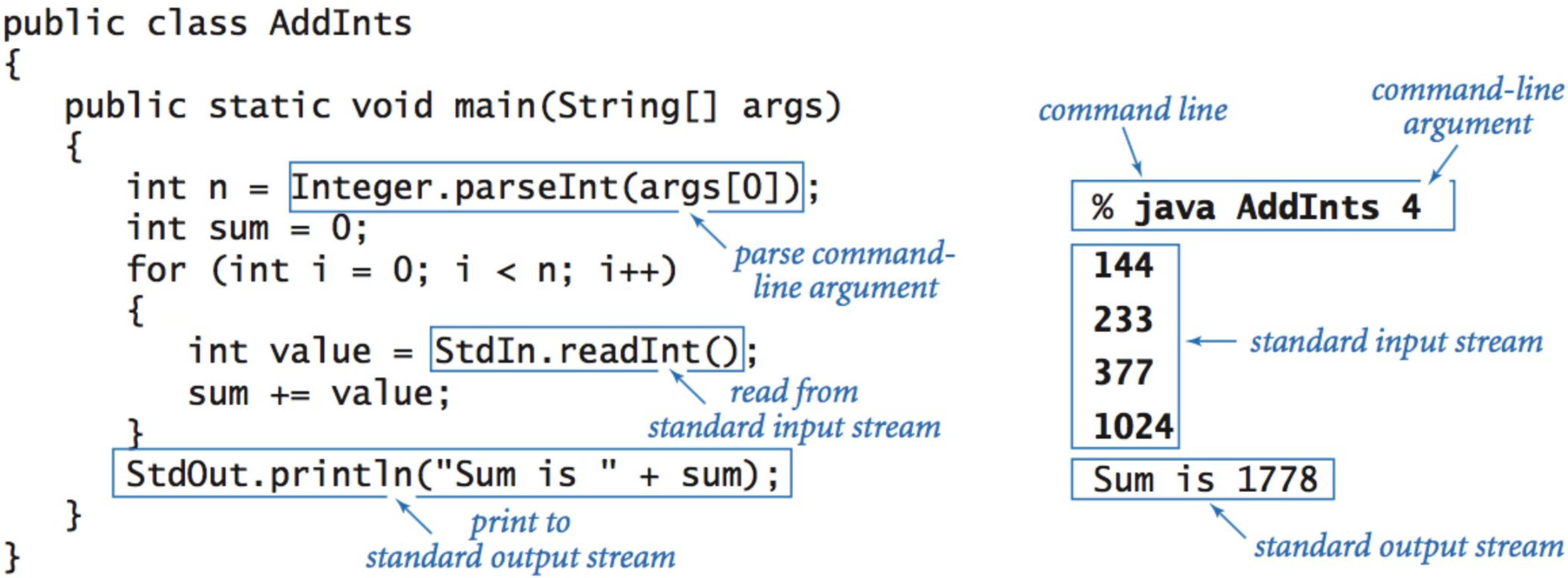
# Array Processing



| | |
|---|---|
| *create an array with random values* | double[] a = new double[n];<br>for (int i = 0; i < n; i++)<br>   a[i] = Math.random(); |
| *print the array values, one per line* | for (int i = 0; i < n; i++)<br>   System.out.println(a[i]); |
| *find the maximum of the array values* | double max = Double.NEGATIVE_INFINITY;<br>for (int i = 0; i < n; i++)<br>   if (a[i] > max) max = a[i]; |
| *compute the average of the array values* | double sum = 0.0;<br>for (int i = 0; i < n; i++)<br>   sum += a[i];<br>double average = sum / n; |
| *reverse the values within an array* | for (int i = 0; i < n/2; i++)<br>{<br>   double temp = a[i];<br>   a[i] = a[n-1-i];<br>   a[n-i-1] = temp;<br>} |
| *copy sequence of values to another array* | double[] b = new double[n];<br>for (int i = 0; i < n; i++)<br>   b[i] = a[i]; |

source: https://introcs.cs.princeton.edu/java/lectures

# Array Processing



```
public class AddInts
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int sum = 0;
      for (int i = 0; i < n; i++)
      {
         int value = StdIn.readInt();
         sum += value;
      }
      StdOut.println("Sum is " + sum);
   }
}
```

parse command-line argument

read from standard input stream

print to standard output stream

```
% java AddInts 4
144
233
377
1024
Sum is 1778
```

command line

command-line argument

standard input stream

standard output stream

source: https://introcs.cs.princeton.edu/java/lectures

# References

https://docs.oracle.com/javase/tutorial/getStarted/problems/index.html

http://sepwww.stanford.edu/sep/jon/family/jos/oop/oop3.htm