CS2050 Technical Documentation

Contents

[**1** **Module 1: Foundation and Review** 3](#_Toc192884445)

[1.1 Set Up Development Environment 3](#_Toc192884446)

[1.2 How to Add an existing file to Eclipse Projects 3](#_Toc192884447)

[1.3 Memory Allocation, Primitive Data Types, Conversion and Casting: 3](#_Toc192884448)

[1.3.1 Memory concept 3](#_Toc192884449)

[1.3.2 Primitive Data 3](#_Toc192884450)

[1.3.3 Numeric Type Conversions 4](#_Toc192884451)

[1.3.4 Casting: 4](#_Toc192884452)

[1.4 Characters and Strings (Manipulation and Operations) 5](#_Toc192884453)

[1.4.1 String and char 5](#_Toc192884454)

[1.4.2 String methods 5](#_Toc192884455)

[1.4.3 Concatenating Strings 5](#_Toc192884456)

[1.4.4 Converting Strings. 5](#_Toc192884457)

[1.4.5 Trim String White Space Characters 6](#_Toc192884458)

[1.4.6 Compare strings 6](#_Toc192884459)

[1.5 Logical Control Structure 7](#_Toc192884460)

[1.5.1 Condition Structure: If/else and Switch 7](#_Toc192884461)

[1.5.2 Iterations: For, While, Do-While 7](#_Toc192884462)

[1.5.3 Nested loop 8](#_Toc192884463)

[1.6 1D Arrays (Declare, initialize, iterate) 9](#_Toc192884464)

[1.6.1 Define: 9](#_Toc192884465)

[1.6.2 Declaring and Create New Arrays 10](#_Toc192884466)

[1.6.3 Pass and Return Arrays: Assign the reference memory address 11](#_Toc192884467)

[1.7 Classes vs. Objects (State and Behavior) 13](#_Toc192884468)

[1.8 Methods, Parameters, and Return 13](#_Toc192884469)

[1.9 Constructors 13](#_Toc192884470)

[1.10 Static vs. Instance Methods and Variables 14](#_Toc192884471)

[Pass by Value and Pass by Reference 15](#_Toc192884472)

[1.11 Encapsulation and Getters/Setters 16](#_Toc192884473)

[1.11.1 Using Encapsulation 16](#_Toc192884474)

[1.12 Inheritance and IS-A Relationships 17](#_Toc192884475)

[1.12.1 Class Relationships 17](#_Toc192884476)

[1.12.2 Inheritance 17](#_Toc192884477)

[1.13 Polymorphism 18](#_Toc192884478)

[1.13.1 Static Binding: Compile-Time Polymorphism (Method Overloading) 19](#_Toc192884479)

[1.13.2 Dynamic Binding: Runtime Polymorphism (Method Overriding) 19](#_Toc192884480)

[1.13.3 Key Benefits of Polymorphism 19](#_Toc192884481)

[1.14 File Input/Output Basics 20](#_Toc192884482)

[1.14.1 File class: 20](#_Toc192884483)

[1.14.2 Steps for Writing to a File 20](#_Toc192884484)

[1.15 Exception Handling, Files and Software Development 20](#_Toc192884485)

[1.15.1 What is Exception Handling? 20](#_Toc192884486)

[1.15.2 Exception Types 21](#_Toc192884487)

[1.15.3 No Try-Catch 21](#_Toc192884488)

[1.15.4 Exception Handling 21](#_Toc192884489)

[1.15.5 Input Data Type Mismatch 22](#_Toc192884490)

[1.15.6 Try-Catch 22](#_Toc192884491)

[1.15.7 Checked vs Unchecked Exceptions 23](#_Toc192884492)

[**2** **Module 2** 23](#_Toc192884493)

[2.1 2D array 23](#_Toc192884494)

[2.1.1 Declare 2D array 24](#_Toc192884495)

[2.1.2 Initialize a 2D Array 24](#_Toc192884496)

[2.1.3 2D array & allocation of memory 25](#_Toc192884497)

[2.1.4 Iterate through the array 2 25](#_Toc192884498)

[2.2 Test-Driven Development (TDD) 26](#_Toc192884499)

[2.3 Sorting algorithms 27](#_Toc192884500)

[2.3.1 Selection Sort: 27](#_Toc192884501)

[2.3.2 Bubble sort 27](#_Toc192884502)

[2.3.3 Insertion sort 28](#_Toc192884503)

[2.4 Searching algorithms 29](#_Toc192884504)

[2.4.1 Linear Search 29](#_Toc192884505)

[2.4.2 Binary Search 30](#_Toc192884506)

[2.5 Arrays Class 31](#_Toc192884507)

[2.5.1 Arrays Class 31](#_Toc192884508)

[2.5.2 Arrays Class Sort and Binary Search Method 31](#_Toc192884509)

[2.6 Algorithm Analysis and Big O 32](#_Toc192884510)

[2.6.1 Algorithm Analysis 32](#_Toc192884511)

[2.6.2 Big-O Notation 32](#_Toc192884512)

[2.6.3 Algorithm Analysis: Insert Sort, Buble Sort and Selection Sort 33](#_Toc192884513)

[2.6.4 Algorithm Analysis: Linear Search Compared Binary Search 33](#_Toc192884514)

# **Module 1: Foundation and Review**

## Set Up Development Environment

[Instruction to set up your environment](https://docs.google.com/document/d/1TDKyQORQsIDRSVgUmC1YPdCkF72GijtB/edit)

## How to Add an existing file to Eclipse Projects

[<https://dzone.com/articles/how-add-existing-files-eclipse>](https://dzone.com/articles/how-add-existing-files-eclipse)

## Memory Allocation, Primitive Data Types, Conversion and Casting:

### Memory concept

A diagram of a person working at a computer

AI-generated content may be incorrect.

### Primitive Data

<https://www.javatpoint.com/java-data-types> This website describes primitive datatypes

* Tells compiler **what type** of data that is stored in a variable
* You must **declare** the type of each variable
  + This means assign a specific type to the variable
* Strongly typed language!
  + This means that once you declare a variable to be a certain it will behave as that type
  + Type safety- you can’t put a floating-point value into an integer unless you explicitly tell it

A diagram of data types

Description automatically generatedA screenshot of a computer

Description automatically generated

### Numeric Type Conversions

The smaller type is converted to a larger type before operation occurs.

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.

2. Otherwise, if one of the operands is float, the other is converted into float.

3. Otherwise, if one of the operands is long, the other is converted into long.

4. Otherwise, both operands are converted into int/short/byte accordingly

### Casting:

Casting is an operation that converts one data type to another

A diagram of casting process

Description automatically generated

|  |  |
| --- | --- |
| **Implicit (Widening**) | **Explicit (Narrowing)** |
| * Casting a value with a smaller range to one with a bigger range * Java does this automatically because safe to do | * Casting a value with a bigger range to one with a smaller range * You must do explicitly because it can result in losing data |
| Example: double d = 3; | Example: int number = (int)3.0; |

## Characters and Strings (Manipulation and Operations)

### String and char

|  |  |
| --- | --- |
| **Char** | **String** |
| *Char is a primitive data type* | *String is NOT a primitive data type; it is an object* |
| *A char variable can hold one single character* | *A String is a sequence of characters* |
| *A char is in single quotes ‘ ‘* | *A String is in double quotes “ ”* |

### String methods

A screenshot of a computer

Description automatically generated

**Character.isDigit(someChar)**

**Character.isLetter(someChar)**

**Character.isLetterOrDigit(someChar) // True if someChar is letter or digit**

**Character.isLowerCase(someChar);**

**Character.isUpperCase(someChar**

### Concatenating Strings

The [Java String class concat()](https://www.javatpoint.com/java-string-concat) method combines specified string at the end of this string. It returns a combined string. It is like appending another string.

A screenshot of a computer code

Description automatically generated

### Converting Strings.

* toLowerCase ()
  + Returns a string that is in all lowercase
* toUpperCase ()
  + Returns a string that is all uppercase

### Trim String White Space Characters

**trim()**

* Returns a string with all white space characters removed from front or end of string
* White space characters include space, tab, line feed, form feed, carriage return,

A white screen with black text

Description automatically generated

### Compare strings

**compareTo(s1)**

Returns an integer:

* + > 0 if string is greater than s1
  + = 0 if string is equal to s1
  + < 0 if string is less than s1

A close-up of a computer code

Description automatically generated

**compareToIgnoreCase**

Return like compareTO but ignoring case considerations

## Logical Control Structure

### Condition Structure: If/else and Switch

A screenshot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| ***nested if*** | ***multi-way if*** |
| *Use when one condition must be true before checking the next condition.*  ***If - else*** | *use when only one condition can be true at a time for the same variable. Here the first condition is tested and if false the next condition is tested*  ***if – else if*** |
| *A screenshot of a computer program  Description automatically generated* | *A white background with black text and yellow text  Description automatically generated* |

### Iterations: For, While, Do-While

|  |  |  |
| --- | --- | --- |
| **While** | **Do-while** | **For** |
| *While* (indefinite loop) loops while some condition is **true**. | *Do-while* (indefinite loop) *same* *while* but the body of a *do-while* loop always executes once. | *For* (definite loop) are used to execute a loop a preset number of times. |
| A diagram of a function  Description automatically generated | *A diagram of a function  Description automatically generated with medium confidence* | *A black screen with text on it  Description automatically generated* |
| **No** ***semicolon*** *at end of loop* | Loop end **with** ***semicolon*** | **No** ***semicolon*** *at end of loop* |
| Declare control variables **outside** *While loop* | Declare control variables **outside** *Do-while loop* | Always declare control variables **inside** *for loop* |
| Using *Increment ++* or *Decrement --* Operators | Using *Increment ++* or *Decrement --* Operators | Using *Increment ++* or *Decrement --* Operators |
| A red circle with green border  Description automatically generated | A green and white rectangle with text and a circle  Description automatically generated | A computer screen shot of a computer code  Description automatically generated |

### Nested loop

You can put while loops in for loops or for loops in while loops.

Loops can be in if/else conditions and if/else conditions can be in loops.

|  |  |
| --- | --- |
| **Nested for loop** | **Nested while loop** |
| Input  A screenshot of a computer code  Description automatically generated | Input  A computer code with text  Description automatically generated |
| Output  A screenshot of a computer  Description automatically generated | Output  A screenshot of a computer code  Description automatically generated |

## 1D Arrays (Declare, initialize, iterate)

### Define:

An **array** = **a data structure** that provides a way to **store more than one value**, but they **must be the same data type**.

Array is **NOT primitive data** typeA diagram of a memory location

Description automatically generated

A diagram of a memory location

Description automatically generated

**Array Defaults**

* The default value is based on the element type the array will store
  + zero for numeric
  + \u0000 - for char type - this is Unicode for "null"
  + false for Boolean type

**Rules for Arrays**

* Once created, an array's size is fixed
* All elements in an array will be the same type
* The array declaration does not allocate memory for the array
* You cannot do anything with array variable numbers until array is constructed with the new operator
* The value stored in numbers is currently null
* After array is created (allocated memory): The value stored in array variable is now the address of the array. A reference variable is used to access the items in an array
* To reference an item in an array on the heap use "bracket" notation
* Once an array is created it has an associated length variable
* Attempting to access an element with an index outside the range of the array is a out of bounds runtime error

### Declaring and Create New Arrays

A close-up of a text

Description automatically generated

### Pass and Return Arrays: Assign the reference memory address

#### Array Assignment

A screenshot of a computer

Description automatically generated

#### Copy Array into New Array: Copy Each Element to New Array Address

A screenshot of a computer

Description automatically generated

#### Passing Arrays to Methods (Passing Arrays By Reference)

A screenshot of a computer

Description automatically generated

#### Returning Arrays from a Method

A screenshot of a computer code

Description automatically generated

#### Comparing two Arrays / reference variables

A screenshot of a computer program

Description automatically generated

## Classes vs. Objects (State and Behavior)

|  |  |
| --- | --- |
| **Class** | **Object** |
| ***A class*** *is a blueprint describing how to build something from the blueprint. It defines an object’s attributes and behaviors.* | ***An object*** *is created using the blueprint of the class. It is an instance of the class* |
| *Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter* | *An object is created with the new operator where memory is allocated on the heap*  *Object must have same name as the class* |
|  | A screen shot of a computer  Description automatically generated |

## Methods, Parameters, and Return

A diagram of a method declaration

Description automatically generated

* **public** - makes method's access public - this means visible to all classes
* **static** - static methods belong to a class – for now make methods static
* **returnType** - data type for value that is returned (int, double, boolean, etc.) or void (nothing to return)
* **methodName** - Meaning full verb, begin with lowercase and use camelCase
* **formal parameter list** - information that is coming into the method
* **Parameters, and Return:**

**parameter and a return value** from main (***data type for value that is returned: int, double, boolean,* etc*.)***

**parameter and no return value** from main ***(void - nothing to return)***

## Constructors

* [Java Constructors](https://www.tutorialspoint.com/java/java_constructors.htm)
* What a constructor is and what a constructor does:

*Special method that creates an object (i.e. instance of a class)*

*A constructor has EXACTLY the same name as the class*

* In what ways is a constructor different from a regular method

*Does* ***not have******a return type*** *- NOT EVEN void*

*Methode is inherited from super class. Constructor* ***is NOT inherited*** *from superclass*

|  |  |
| --- | --- |
| **default constructor** | **overloaded constructor** |
| *If you do not create any constructor in the class, Java provides a default constructor that initializes the object.*  *It takes no arguments*  *It has an empty body - no code*  *It does nothing to the instance variables* | *Constructor overloading means multiple constructors in a class. When you have multiple constructors with different parameters listed* |

* Understand how constructors are invoked using new and what is happening in memory on the stack and on the heap

*On main, when Simplerectangle is invoked, it moves to the SimpleRectangle constructor on SimpleRectangle class*.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

*Then rectangle1 object was created on the HEAP*

A screenshot of a computer

AI-generated content may be incorrect.

## Static vs. Instance Methods and Variables

|  |  |
| --- | --- |
| **Instance(= object) Variable and Methods** | **Static(=Class) Variable and Methods** |
| Instance Variable   * A variable that is part of an instance of a class, as opposed to a class variable that is part of the class itself. * State of an object - things an object knows about itse | Static (Class) Variable   * A variable that is attached to a class rather than to an instance of the class. * Fields of a class for which one, and only one, copy exists, regardless of the number of instances of the class * Constants of a class are shared by ALL objects of a class * Constants by default use static keyword! |
| Instance Method   * A method that is part of an instance of a class * These methods can be executed only through a reference to an instance of a class * Behavior of an object - things an object can do | Static (Class) Method (declare with **static** keyword)   * A method that is attached to a class rather than to an instance of the class * Methods declared as static are class methods * Methods of a class that can be executed without the need to reference a particular instance of the class * Example: The Math class methods are all static! Math.sqrt(), Math.sin(), Math.pow(), etc * You can use the object reference but this not the best way to call the method |
| Instance variables and instance methods belong to each object’s specific instance | Not necessary to have an instance of the class to access this method |

The different types of variables in a Java program are:

* Local variables - declared in a method
* Instance variables - declared within a class
* Static variables - declared within a class

Scope

* Local variable - starts from its declaration and continues to the end of the block that contains the variable, such as
  + declared in for loop {block of code}
  + declared in method {block of code}
* Instance variables and static variables scope is the entire class.

## Pass by Value and Pass by Reference

|  |  |
| --- | --- |
| Pass by value | Pass by Reference |
| means a copy of the actual value is passed.  Primitive data types are passed by value | means a memory address is passed that refers to where values are stored on heap.  An array, object are passed by Reference |
| A white cup with orange and blue text  Description automatically generated with medium confidence | A white cup with orange and black text  Description automatically generated with medium confidence |

## Encapsulation and Getters/Setters

**Visibility Modifiers**:

* **Public**
  + Indicates that the class, method or variable can be accessed from ANY other classes
  + Can apply to class or members of a class (variables/methods)
* **Private** 
  + Indicates the method or variable can be accessed ONLY from within its own class
  + Can apply only to members of a class (variables/methods)

**Encapsulation**

**A diagram of a pill

Description automatically generated**

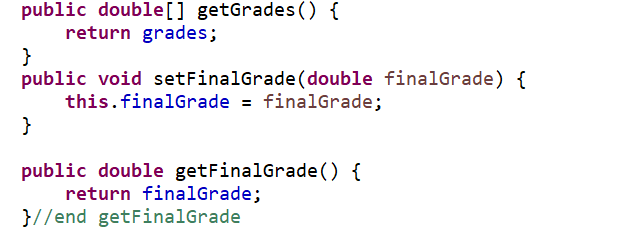
### Using Encapsulation

* **Add the** **private** modifier to all instance variables

A close up of a text

Description automatically generated

* **Create** **getters and setters** to **retrieve and set** private instance variables
  + getter - method that returns the value of an instance variable
  + setting - method that sets the value of an instance variable



* Keyword **this** is the name of a **reference that refers to the object itself** when writing code in the class.

## Inheritance and IS-A Relationships

### Class Relationships

* Association:
* Aggregation:
* Composition:
* Inheritance

*Source* <https://medium.com/@humzakhalid94/understanding-object-oriented-relationships-inheritance-association-composition-and-aggregation-4d298494ac1c>

### Inheritance

A diagram of a child class

Description automatically generated

*Image source :* <https://www.tutorialkart.com/java/inheritance-in-java/>

Subclass/child class **extends** from super/father

**Inherit:** **Data fields/variables, Methods**

**Not inherit**: Constructors

Example: **subclass student**, inherited from super class person: first name, last name, email and methods like getFirstName, getLastName, getEmail

A close up of text

Description automatically generated

**Constructors are NOT inherited by the subclass**

* This means you cannot directly call a superclass constructor
* But superclass constructor can be invoked explicitly or implicitly.
  + To access explicitly - use the *super* keyword
  + To access implicitly (automatically) - If keyword *super* is not explicitly used, the superclass's default constructer is automatically invoked.

**Superclass Methods**

* The **super** keyword can also be used to access a method in the superclass (parent)
* This is NOT necessary to do in most cases

A screen shot of a computer

Description automatically generated

* It IS NECESSARY when you override a superclass (parent) method in the subclass (child)
  + If you want call overridden method in subclass, leave super off.
  + If you want to call the method in the super class, need to use super keyword.

## Polymorphism

Polymorphism is the process of defining same method with different implementation. That means creating multiple methods with different behaviors.

*A yellow rectangles with black text

AI-generated content may be incorrect.*

Polymorphism is one of the key principles of object-oriented programming that allows methods to be used in multiple ways. There are two main types: [Static Binding (method overloading) and Dynamic Binding(Method overriding) in Java](https://www.javatpoint.com/static-binding-and-dynamic-binding)

### Static Binding: Compile-Time Polymorphism (Method Overloading)

Occurs when multiple methods share the same name but differ in their parameters (number, type, or both).

Determined at compile time

*A screenshot of a computer program

Description automatically generated*

### Dynamic Binding: Runtime Polymorphism (Method Overriding)

Occurs when a subclass provides a specific implementation of a method that is already defined in the superclass.

The overridden method is determined at runtime based on the actual object type.

A screen shot of a computer code

Description automatically generated

### Key Benefits of Polymorphism

✔ Code Reusability – Avoids code duplication by using generalized methods.

✔ Scalability – New types can be added without modifying existing code.

✔ Maintainability – Code remains easy to read, manage, and extend.

✔ Flexibility – A single interface can be used for multiple implementations.

## File Input/Output Basics

### File class:

A white paper with black text

Description automatically generated

### Steps for Writing to a File

A screenshot of a computer program

Description automatically generated

## Exception Handling, Files and Software Development

### What is Exception Handling?

* + Ensures program stability by managing errors gracefully.
  + Prevents unexpected crashes and provides useful error messages.

### Exception Types

Exceptions are objects and the root class for exceptions is java.lang.Throwable

Java provides a number of predefined exception classes

* Error
* Exception
* RuntimeException
* ClassNotFoundException
* NullPointerException
* ArithmeticException
* Etc.…

User defined exceptions

* Created by extending the Exception class or a subclass of Exception

### No Try-Catch

When no exceptions occur: If code contains try-catch blocks, code in try blocks execute and catch blocks are skipped

When an exception occurs, here is what happens:

If offending code is not embedded in a try-catch block (not caught in current method)

* Exception is passed to calling code
* If the caller did not embed the method call in a try-catch, the above step is repeated until
  + Exception is caught or
  + Exception is passed to main
* If reach main and if the call is not embedded in a try-catch block
  + Program halts (crashes)
  + Trace of the method calls, the exception type, and its error message is printed

### Exception Handling

When a program is able to continue execution when a runtime error occurs

* Runtime errors throw exceptions
  + division by 0
  + user enters a double when code expects an integer
  + array index out of bounds
* To use exception handling in your program , include try-catch blocks

A black text on a white background

Description automatically generated

Java **try** block is used to enclose the code that might throw an exception.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute.

The try block must be followed by either catch or finally.

Java **catch** block is used to handle the Exception by declaring the type of exception within the parameter.

### Input Data Type Mismatch

If you enter a letter for a grade where it expects a double

* What happens if you enter a double when the data type is an integer?
* What happens if you enter a non-numeric character for numeric data type?

A screen shot of a computer code

Description automatically generated

Type input.nextDouble and look at the methods API (Application Program Interface). Not only does it tell you what it does, parameters and returns but what exception can occur

### Try-Catch

If offending code is embedded in a try-catch block

* Java searches the catch blocks, in order, for matching exceptions.
  + The search process starts in current method
  + If no matching catch in current method, then works backwards through method calls to find the matching exception handler
  + At each point, Java determines if the type of the exception object that was thrown is an instance of the exception in the catch blocks
* The first matching catch is executed to handle the exception.
* After first matching catch is executed, the program restarts after the try-catch block.
  + The rest of the try block is skipped
* If no exceptions occur in a try-catch block, the catches are skipped.
* An optional finally clause can be placed after ALL catches.
  + It is executed whether an exception occurs or not.

### Checked vs Unchecked Exceptions

|  |  |
| --- | --- |
| Unchecked Exceptions | Checked Exceptions |
| RuntimeException and Error and their subclasses are known as unchecked exceptions  Generally unrecoverable programming logic errors  Compiler does not force programmer to catch  Example:   * A for-loop that accesses an array element out of bounds will cause an ArrayIndexOutOfBoundsException   In this case, the for-loop code should be fixe | Any exception that is NOT an Error or RuntimeException  Compiler forces the programmer to check and deal with exception   * + Try-catch block   + Declare in method |

# **Module 2**

## 2D array

A 2D array is an array of arrays

To get the **number of rows** in array is use **arrayName.length**

To get the **number of columns** use **arrayName.[0].length** where 0 can be any row

A diagram of a number

AI-generated content may be incorrect.

### Declare 2D array

A diagram of a number of rows

AI-generated content may be incorrect.

Example:

A math equations with text

AI-generated content may be incorrect.

### Initialize a 2D Array

#### Initialize with nested for loop

Fill 2D Array Cars with car object:

Using nest for loop to access every position of cars array: outer loop through each row; inner loop iterates through each column in the current row.

Construct *car object* with specific model.

Add *car object* to specified [row][col] position in the *cars array.*

A screenshot of a computer code

AI-generated content may be incorrect.

Fill 2D table array with multiplication values:

A table of multiplying equations

AI-generated content may be incorrect.

#### Initialize with a list



### 2D array & allocation of memory

A screenshot of a computer

AI-generated content may be incorrect.

### Iterate through the array 2

A computer code with text

AI-generated content may be incorrect.

A computer code with black text

AI-generated content may be incorrect.

## Test-Driven Development (TDD)

Test-driven development refers to an Agile methodology to approach programming in which three activities are tightly interwoven: testing (in the form of writing [unit tests](https://www.agilealliance.org/glossary/unit-test/)), design, coding and refactoring.

Following TDD helped us build quality software, by reducing errors and improving testability and maintainability.

* Break the problem into smaller parts
* Design test cases to help understand logical thinking
* Implement each algorithm and test before implementing next step.
* [Refactoring](https://refactoring.guru/refactoring) - remember [Single Responsibility Principle (SRP)](https://dev.to/eidorianavi/functions-and-the-single-responsibility-principle-48ae)  and [Don’t Repeat Yourself (DRY](https://dzone.com/articles/software-design-principles-dry-and-kiss) )

Example: bubble sort

Break the problem into smaller parts: Bubble sort is also called sinking sort. Smallest values bubble to the top, largest values sink to the bottom:

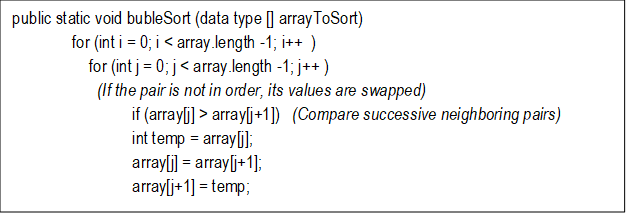
* + Compare successive neighboring pairs
  + If the pair is not in order, its values are swapped
  + If the pair is in order, the values remain unchanged
  + Make several passes through array until sorted

Design test cases:

A table with numbers and symbols

AI-generated content may be incorrect.

Design Algorithm Based on task and test cases:



## Sorting algorithms

### Selection Sort:

* Sorting algorithm in which each value, one at a time, is placed into its final sorted position in the list.
* Reference link [The Selection Sort](https://runestone.academy/ns/books/published/javads/sort-search_the-selection-sort.html?mode=browsing)
* Animation link [Selection Sort](https://liveexample.pearsoncmg.com/liang/intro12e/html/SelectionSort.html)

#### Selection Sort Algorithm:

* Finds the smallest element in the unsorted part and swaps it into place.

Step-by-step:

* 1. Find the smallest element and swap it with the first element.
  2. Move to the next position and repeat for the rest of the array.
  3. Repeat until sorted.
* Efficiency: Always runs in due to repeated comparisons, making it inefficient for large datasets

#### Selection sort code

A screen shot of a computer code

AI-generated content may be incorrect.

### Bubble sort

* Sorting algorithm in which values are repeatedly compared to neighboring elements in the list and their positions are swapped if they are not in the correct order relative.
* Reference link [**The Bubble Sort**](https://runestone.academy/ns/books/published/javads/sort-search_the-bubble-sort.html?mode=browsing)
* Animation [Bubble Sort Animation by Y. Daniel Liang](https://liveexample.pearsoncmg.com/dsanimation/BubbleSortNeweBook.html)

#### Bubble Sort Algorithm:

* Bubble sort is also called sinking sort. Smallest values bubble to the top, largest values sink to the bottom
* Make several passes and on each pass the largest value is moved into its final position

Step-by-step:

1. Compare each pair of adjacent elements and swap if they are in the wrong order.
2. Pass through the list multiple times until no swaps are needed.
3. Each pass moves the largest element to its correct position

#### Bubble sort code

A computer code with text

AI-generated content may be incorrect.

### Insertion sort

* Sorting algorithm in which each value, one at a time, is inserted into a sorted subset of the entire list
* Reference link [The Insertion Sort](https://runestone.academy/ns/books/published/javads/sort-search_the-insertion-sort.html?mode=browsing)
* Animation [Insertion Sort Animation](https://liveexample.pearsoncmg.com/dsanimation/InsertionSortNeweBook.html)

#### Insert Sort Algorithm:

* Inserts each element into the correct position within the sorted part of the array.

Step-by-step:

* 1. Start with the first element as sorted.
  2. Take the next element and shift it left until it’s in the correct position.
  3. Repeat for all elements.
* Efficiency:
  + Best Case (): If the array is already sorted, only one comparison per element.
  + Worst Case (): If the array is sorted in reverse order, each element moves across the entire sorted part.

#### Insert sort code

A screen shot of a computer code

AI-generated content may be incorrect.

## Searching algorithms

### Linear Search

* Search algorithm in which each item in the list is compared to a specific key starting at the beginning and comparing the key with each element sequentially
* Linear search is good for:

Small arrays

Unsorted arrays

* [LinearSearch](https://liveexample.pearsoncmg.com/liang/intro12e/html/LinearSearch.html) code animation
* Referent link [Linear Search in Java](https://www.javatpoint.com/linear-search-in-java)

#### Linear Search Algorithm:

Starting at beginning of an array and looking for a specific value one element at a time

When a match is found that index of the match is returned, otherwise a -1 is returned

The elements in the array are in any order

A number grid with red and green squares

AI-generated content may be incorrect.

* Not very efficient algorithm - execution time grows linearly as the array size grows!
  + Best case: 1st element is key we are looking for
  + Worst case: last element is key or doesn't find key
  + Average: 50% of elements are examined before key is found

#### Linear Search code

A screen shot of a computer code

AI-generated content may be incorrect.

### Binary Search

* Searching a pre-sorted array by splitting the array in half on each comparison.
* [Binary search](https://liveexample.pearsoncmg.com/dsanimation/BinarySearcheBook.html) Visualization

#### Binary Search Algorithm:

Starts with element in the middle of the array

* + If key is less than "middle" element, then only search for key in 1st half of array
  + If key is equal to "middle" element, return match!
  + If key is greater than "middle" element, then only search for key in 2nd half of array

When a match is found that index of the match is returned, otherwise a -1 is returned

More Efficient algorithm - array shrinks by 50% after each comparison!

#### Binary Search code

A screenshot of a computer program

AI-generated content may be incorrect.

## Arrays Class

### Arrays Class

* There are many tasks that are useful when working with arrays. Java provides an Arrays class that includes **static methods** for searching, sorting, filling, comparing, etc.

A close-up of words

AI-generated content may be incorrect.

* Static methods:

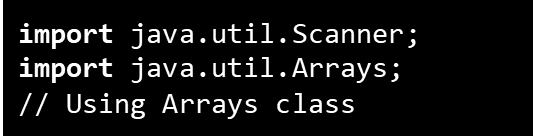
Allow you to call the methods in a class without creating an object of the class

Are NOT associated with a specific object of the class

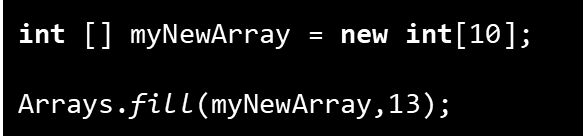
To call static methods you use the class name

These are similar to the static methods you create in your class

* Import Arrays Class



* Arrays Class allows you to call the static methods without creating an Object



### Arrays Class Sort and Binary Search Method

**Array.sort:**

A screenshot of a computer

AI-generated content may be incorrect.

**Array Search:**

A screenshot of a computer

AI-generated content may be incorrect.

## Algorithm Analysis and Big O

### Algorithm Analysis

Algorithm Analysis is a Topic in computer science where you examine the efficiency of algorithms

* Predict the efficiency of algorithms
* Efficiency covers lots of resources: CPU (time) usage, CPU (time) usage, Memory usage, Disk usage, Network usage
* Will be looking only at CPU usage – time complexity – as size of input increases
* Common algorithmic techniques for developing efficient algorithms: Dynamic programming, Divide-and-conquer, Backtracking

Purpose

* Possible to have several algorithms that are a solution to a problem. Need to compare the efficiency of algorithms to select the best algorithm
* Understanding algorithm efficiency is also important when building data structures
  + Data structure built correctly
    - Efficient use of both CPU and memory
    - Can be reused in many different applications
  + Data structure built incorrectly
    - Problematic in terms of CPU and memory
    - Not reusable

### Big-O Notation

* Big-O Notation estimates algorithm time complexity
  + Classifies the upper-bound (worst case) time behavior of an algorithm
  + Estimates change in execution time based on input size
  + Gives us a growth rate – use to compare algorithms
* The rating (Big-O) tells you how fast you can expect the algorithm to be as the number of inputs gets larger and larger.

#### Growth Rates

* [Big-O Cheat Sheet](http://www.bigocheatsheet.com/) contains the following helpful chart. As n increases, the various growth functions diverge dramatically.

A diagram of a graph

AI-generated content may be incorrect.

### Algorithm Analysis: Insert Sort, Buble Sort and Selection Sort

|  |  |  |
| --- | --- | --- |
| Insert Sort | Selection Sort | Buble Sort |
| Small inputs, all three were "about the same | | |
| Best performance, especially for small/nearly sorted arrays. | Middle performance, but does unnecessary swaps, making it slower than Insertion Sort. | Worst performance, due to excessive swaps |
| Efficiency:  Best Case (): If the array is already sorted, only one comparison per element.  Worst Case (): If the array is sorted in reverse order, each element moves across the entire sorted part. | Efficiency: Always runs in due to repeated comparisons, making it inefficient for large datasets | Efficiency:  Best Case (): If already sorted, only one pass is needed.  Worst Case (): Requires multiple swaps per element.  Rarely used due to inefficiency |
|  |  |  |

### Algorithm Analysis: Linear Search Compared Binary Search

|  |  |
| --- | --- |
| Linear Search | Binary Search |
| Not very efficient algorithm | More Efficient algorithm - array shrinks by 50% after each comparison |
| Time complexity: O(n) | Time complexity: O(log2 n) |
|  | |