CS2050 Technical Documentation

Contents

[**Module 1: CS 1050 Review Concepts** 2](#_Toc190459236)

[Set Up Development Environment 2](#_Toc190459237)

[How to Add an existing file to Eclipse Projects 2](#_Toc190459238)

[Primitive Data types Conversion and Casting: 2](#_Toc190459239)

[Numeric Type Conversions 3](#_Toc190459240)

[Casting: 3](#_Toc190459241)

[Characters and Strings (Manipulation and Operations) 4](#_Toc190459242)

[String and char 4](#_Toc190459243)

[String methods 4](#_Toc190459244)

[Concatenating Strings 4](#_Toc190459245)

[Converting Strings. 5](#_Toc190459246)

[Trim String White Space Characters 5](#_Toc190459247)

[Compare strings 5](#_Toc190459248)

[Classes vs. Objects (State and Behavior) 5](#_Toc190459249)

[Constructors 6](#_Toc190459250)

[Methods, Parameters, and Return 7](#_Toc190459251)

[Pass by Value and Pass by Reference 7](#_Toc190459252)

[Control Flow Statements (Conditionals) 8](#_Toc190459253)

[1D Arrays (Declare, initialize, iterate) 8](#_Toc190459254)

[Define: 8](#_Toc190459255)

[Declaring and Create New Arrays 8](#_Toc190459256)

[Pass and Return Arrays: Assign the reference memory address 9](#_Toc190459257)

[Encapsulation and Getters/Setters 11](#_Toc190459258)

[Loops 12](#_Toc190459259)

[For, While, Do-While 12](#_Toc190459260)

[Nested loop 13](#_Toc190459261)

[Inheritance and IS-A Relationships 13](#_Toc190459262)

[Class Relationships 13](#_Toc190459263)

[Inheritance 13](#_Toc190459264)

[Polymorphism 14](#_Toc190459265)

[Static Binding: Compile-Time Polymorphism (Method Overloading) 15](#_Toc190459266)

[Dynamic Binding: Runtime Polymorphism (Method Overriding) 15](#_Toc190459267)

[Key Benefits of Polymorphism 15](#_Toc190459268)

[File Input/Output Basics 16](#_Toc190459269)

[File class: 16](#_Toc190459270)

[Steps for Writing to a File 16](#_Toc190459271)

[Static vs. Instance Methods and Variables 17](#_Toc190459272)

[Instance Variable and Methods 17](#_Toc190459273)

[Static Variable and Methods 17](#_Toc190459274)

# **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)

## Primitive Data types Conversion and Casting:

<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

## 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 code  Description automatically generated | A screen shot of a computer  Description automatically generated |



Blueprint of a house with a few floors

Description automatically generated

## Constructors

* [Java Constructors](https://www.tutorialspoint.com/java/java_constructors.htm)

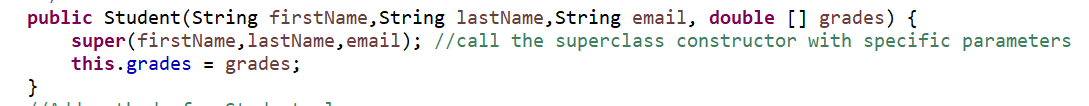
A black screen with white text

Description automatically generated

|  |  |
| --- | --- |
| **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* |

**Example:**

Constructor of student class:



Create student object using constructor



## 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)***

## 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 |

## Control Flow Statements (Conditionals)

## 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** type

A diagram of a memory location

Description automatically generated A diagram of a memory location

Description automatically generated

### 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

## 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**

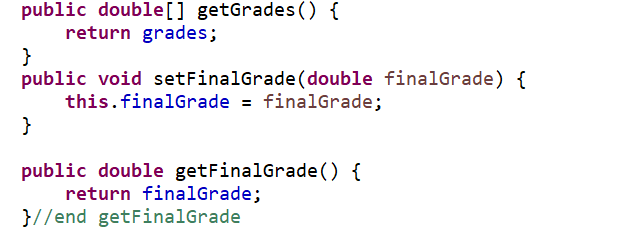
**Using Encapsulation**

* **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.

## Loops

### 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 |

## 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.

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.

## Static vs. Instance Methods and Variables

|  |  |
| --- | --- |
| Instance Variable and Methods | Static 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   * 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 |

## 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 |