CYDEO

Java Terms

Binary

- A numbering system (language) that computers understand
- Made of 0s and 1s to represent two states. Computers use the combination of these states to communicate with electric signals
- In terms of memory: one binary number is a bit, and a byte is made of 8 bits [these are memory units]
- A character or a number needs to be representing in some way. A combination of binary digits can represent them
- Each bit in a byte can represent the power of 2
- For characters see ASCII.
- Number we use are decimal numbers

Ex:

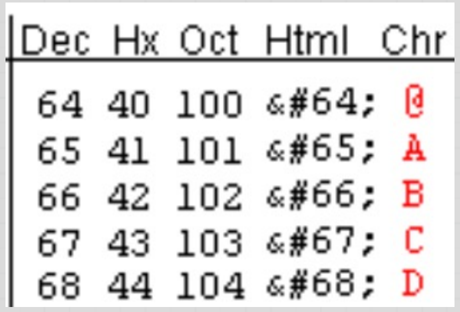
```
00000000 -> 0
00000001 -> 1
00000010 -> 2
00000011 -> 3
00000101 -> 5
10000000 -> 128
11111111 -> 255
```

ASCII: American Standard Code for Information Interchange

- A computer can only understand binary, so what about characters

- Each character has a number associated with it. Those numbers are done read in their binary composition for the computer to understand which characters we are using

- The ASCII focuses on only common English characters and symbols, but other characters, such as a character from a different language, or even emojis have a number associated with them. That number is called a Unicode. So Ascii is only a small part of the Unicode system



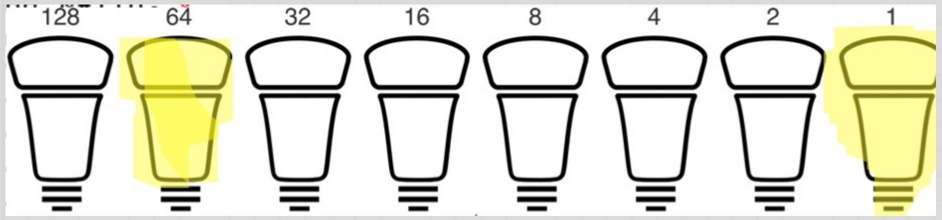
Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@
65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D

Source: https://www.asciitable.com/

Example: How is 'A' represented

'A' has a decimal number of 65

65 In binary is: 01000001



Algorithm

- An algorithm is a step-by-step solution to a problem
- Classic example: Phone Book
- In programming: Algorithms are used to define common functionality for everyone to use

Pseudocode

- A representation of the algorithm/code in a more understandable language
- Using English, it is possible to explain the steps taken by the code without having to understand each aspect of the code
- It is not ”real” code, but can help explain how the code is structured
- Ex:

```
Ask for the age
check if the age is less than 18
    less than 18 return invalid
    more than or equal to 18 return eligible
```

High Level / Low Level

- Computers understand binary, but we do not speak in 0s and 1s. Our languages are different

- We use high level languages, which are human understandable, to write code. Even a programmer doesn’t have to only speak in low languages for the computer to understand. That’s what programming languages are for

- Low level languages are what machines/computers understand

- Programming languages vary on the scale between high and low level, but they are meant for us to be able to write code in a language understandable to us and will eventually be translated to the computer level. Java is a high-level language

IDE: Integrated development environment

- Code can be written in a text editor. Then there is other steps to translate the code and run it
- Even though it is possible to write and work with code in a normal text editor there is tools created for programmers to make it easier to write code.
- An IDE: Integrated development environment is a tool that can help to write code, translate and run code easier
- We will use IntelliJ as our IDE. It is the most popular java IDE
- Besides the base functionality IDEs also offer things like shortcuts, control of the editor, generation of common code fragments

IntelliJ Components

- When working in an IntelliJ project there is some sections to become familiar with:
- src: the source folder is where code is written
- package: a package acts like a folder for code in the src
- console: the window that shows the output of our code (opens at the bottom by default)
- editor: the window that opens the code file, allowing viewing and editing of the file (opens at the right by default)
- out: a folder for bytecode(translated java files). You won’t need to interact with this folder

Syntax

- The syntax is the correct rules or grammar of the code
- We will learn syntax of components as we go through each topic
- Some example of basic syntax:

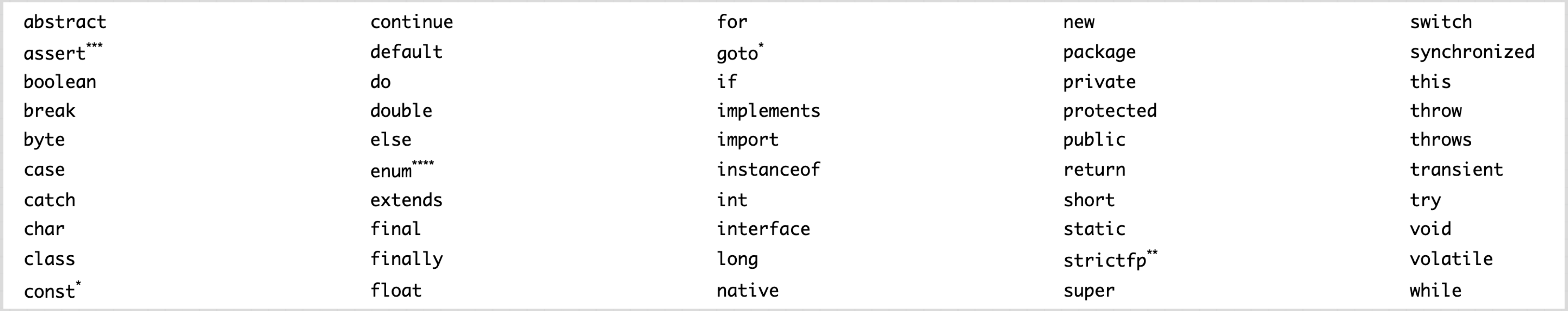
  ; → A semi colon is used to end a statement
  {} → curly brackets are used to group code blocks
  () → parenthesis are needed with certain code

- Giving invalid syntax means the program will not work

Java Reserved Words

- There are some keywords used in the java syntax for different purposes. You will not be able to use these words in things like **variables** because they already exist in the language.

- No, you don't need to memorize these, we are just emphasizing what these words are. As we go through the course, we will understand what most of them are doing



abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

Source: oracle documentation(https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html)

Indentation

- Indenting is extra spaces in the beginning of the line. (Often given with a tab input)
- Indenting is important in programming because we want our code to be readable.
- Java is not space sensitive, so spaces are not syntax we need to worry about. The code will work as intended, but it will not be readable to ourselves or others

Incorrect Indentation:

```
class Hello {
public static void main(String [] args){
System.out.println("Hello World");
}
}
```

Incorrect Indentation:

```
class Hello { public static void main(String [] args){ System.out.println("Hello World");}}
```

Correct Indentation:

```
class Hello {
    public static void main(String [] args){
        System.out.println("Hello World");
    }
}
```

Comments

- Comments are only text. They are not code. They do not get executed
- If comments are not code, we do we need them? Comments are a crucial part of programming. They allow us to define the main functionality of the code
- Others can read our comments to understand how to use our code. Or we can read our comments later to remind ourselves what the code was doing.
- We can put reminders. TODO comments
- Components are part of documentation too. The oracle developers make comments to explain part of the existing java code

Execute / Run

- During execution, the code is read one line a time, from top to bottom & left to right. Each line is run, meaning the functionality of each line is performed. Each line could include different functionality such as operators being used, methods being called, or many other code statements.

- We run code using a main method. There is other ways to execute code, but the main method is the first approach we use before automation.

- IDEs, like IntelliJ offer a quick way to execute the code with a button press, otherwise the code would manually need to be execute with commands

Compile

- Java code needed to compile in order to work. Compiling a java file means to translate it into a bytecode file. The bytecode file can then be interpreted by the JVM.
- Human understandable files with java code need to be translated into a machine understandable file
- Another benefit of IDEs, like IntelliJ, is the automatic compiling. When we write code the tool is able to help us determine of the program will compile, usually meaning if the syntax is all correct.
- If we wanted to compile and execute manually it would take multiple more steps. Each new change would require the file to be compile again, then executed

Convention

- Conventions are commonly used approaches for different parts of the code.

- Conventions are not the same as syntax. Syntax is something that much be correct otherwise the program will not work. If convention is not followed the program can still run.

- Conventions are common principles all programs follow. Each different concept has its own conventions and will be discuses as new topics are brought up. Conventions should be followed because they make it easier for anyone to read the code and work alongside you.

- Example: packages in java are created with lowercase letters. If uppercase letters were given the program would still compile, but the convention is to define them with lowercase letters only.
- Another Example: Classes are defined with the first letter being uppercase

Declare / Assign / Reassign

- Declaring a variable or reference means to set up a container that is ready to use. There is not value yet.

- Assigning a variable or reference means to give the actual value. If it's a primitive datatype the given to the variable. If it is an object type the object is assigned to the reference. (You can think of everything as a variable for now)

- Reassigning is done when there is already a value in the variable, but a new value is assigned anyway. The old value is lost

Concatenation

- Concatenation allows multiple String types to be combined into one.
- If any other type is concatenated with a String type the result of that combination is a String type
- Concatenation is done with the + operator
- Q: How to tell the difference between addition and concatenation? Both use the + operator, but to determine which one is being done the datatypes of the value needs to be checked. If both values are number types, it is addition, but if even one type is a String, then it will be concatenation

Increment

- Incrementing is updating a value by 1 –or- adding 1 to the value
- Updating a number by one is a very common function in programming. It is used to count or used in loops to keep track of iterations(cycles)

Decrement

- Decrementing is reducing a value by 1 –or– subtracting 1 from the value
- Decrementing is not as common as incrementing, but can be useful is certain situations
- Common uses: reversing logic

Hard Coding

- Hardcoding means to assign value directly in the code
- Hardcoding is something to avoid in most cases.
- The opposite would be dynamic, when the value of something is coming from outside of the code (could be from a different file like text file, excel file, or even different source like browser or database)
- Imagine you are gathering information about a person’s name. The name would be hardcoded if you typing “James” directly into the code instead of dynamically getting the information from outside the code

Debugging

- Debugging is an important part of the working with code.
- Debugging is going through the code manually to determine each steps' functionality
- Debugging is often done when the code is not behaving as expected. By checking the lines one at a time its possible to determine where the problem, or bug, is occurring and that can allow us to fix
- IntelliJ has a debugging feature that gives us control to execute one line at a time and show the result of each line

Reusability

- Having any repeating code is not recommended. Having repeating code does not cause a syntax issue but will go against preferred conventions
- To avoid repeating ourselves we want to create code that is reusable, meaning we will not need to repeat the code anywhere again and just use the code again
- Reusable code also allows the block of statements to be used easier in other places, such as other files

Readability

- Reading code is an important part of programming because a lot of the time you will be using code that is created from someone else.

- Writing code in a readability way will ensure anyone who looks at our code will understand it. Even reading your own code back at a later point will be easily when the code is created in a readable way.

- How can code be readable? It's done by following common conventions and practices. Things like following camel case convention for variable names or indenting certain blocks. Readability practices for the topics will seen as the course goes on

Code Block

- A code block is a group of code statements. Most of the code blocks are grouped in sets of curly brackets.
- Curly brackets group the statements for topics such as methods, loop, etc. The different grouping of code statements will be seen as these topics are coming up
- One very common mistake made early on is writing the code in the wrong spot. The code may have been needed in the specific code block, but it may have been written outside the bracket and that could cause the code not to work

DRY: Don’t Repeat Yourself

- DRY is a common principle followed in programming to avoid repeating ourselves
- The idea is to not write the code statements multiple times, but instead writing reusable code that can be used multiple times.
- Any time you are writing code that is doing the same thing as something else there should be an evaluation of if you need to use the code written before, or how you can improve the reusability
- The concept of loops is all about not repeating yourself

Documentation

- Documentation is like a manual or instructions of how to use the code that was written by someone else.
- In programming whenever you use a new technology the best way to learn how to use it and the best reference of how the technology works the documentation should be checked.
- If something about java needs clarification the best resource will be the official documentation giving by oracle to explain how everything works

  https://docs.oracle.com/en/java/javase/11/docs/api/

Immutable

- Mutability is to change the value after creation, but immutability is the opposite, after creation the value cannot be changed
- Immutability of objects is first seen with the String class. After a String object is created the value of it cannot be changed. It is fixed. Details about what this means will be talked about in class

Index

- There is a couple topics that use indexes. An index is just a number that is related to some value
- In Strings the index is position of characters in the String or in data structures the index if the position of the element
- Using the indexes will be talked about in more details related to each topic that uses them

Method

- A block of code statements that has some functionality.
- Methods are created to improve reusability, readability and maintainability
- We will create methods as well as using methods created by others. The main things to understand about a method are the parameters(inputs), the return type (output), and what the method does

Parameter / Argument

- A parameter act as inputs for a method. When a method is created it is possible to define certain variables that are used within the method. Parameters are local variables to the method

- An argument is the actual value given when calling the method. When you want to use a method, if there was a parameter defined it means an argument must be provided matching the defined datatype for the program to compile and run

Return

- A method give be either void or have a return type
- The return is some value passed from the method to where it was called. It is like the output of the method in a way.
- A method can define a return type and return a single value.
- When a return statement is executed the method stops and that value is passed to where the method was called

Overloading

- Method overloading is having a method with the same name, but different parameters
- Overloading allows a method to share name but also allow slightly different code blocks to run based on the given arguments.
- Overloading improves the reusability and readability of the code
- Overloading is used often. Ex: print statement

Call / Invoke

- Calling and Invoking are used in the same context of a method, but there is a small difference
- Calling a method is said whenever you write code to execute a specific method
- Invoking is also about executing a specific method, but it is usually used in the context that the method was called automatically instead of you writing a code to specifically run that method

Iteration

- An iteration is one cycle/execution of the code block defined in a loop
- When a loop is running there is usually multiple times the code is meant to be run, each execution is an iteration
- A loop that runs for 5 times has 5 iterations of that code

Data Structure

- A data structure is collection of data into one unit instead of having multiple different variables
- There is always a large amount of data needed in programming and it would not be productive to interact with that data one piece at a time. Data structures are needed to work with multiple values
- Arrays and Collection types are the data structures used in java

Element

- Elements are a single value of a data structure
- For example, element is the word used to reference the values that are stored in an array.
- Uses: “The first element is…”, “the second element….”, “Iterating through all elements”

Utility Class

- Utility classes have a group of useful methods.
- There are existing utility classes, like Arrays or Collections which have useful methods for those topics
- Also, it is common for a programmer to make a utility class for themselves to store methods that can help them in their workflow.
- You will create utilities in automation

Wrapper Class

- Wrapper classes are a group of classes that were created as object representations of the primitive datatypes
- These classes also have some useful variables and methods related to those datatypes.
- For example, the Integer class has the max and min values that the datatype can hold, and the class has a parse method that is very helpful to convert a String type to a number
- The great part of wrapper classes are the automatic conversion that occurs when going from primitive types to object types and vice versa

Autoboxing / Unboxing

- Autoboxing is when a primitive datatype is converted to a wrapper class object
- Unboxing is when a wrapper class object is converted to a primitive datatype
- Both autoboxing and unboxing happen automatically

Class

- A class is a blueprint for objects
- Classes are defined in .java files and objects are created from a class
- You can define information (variables) and actions (methods)

Object

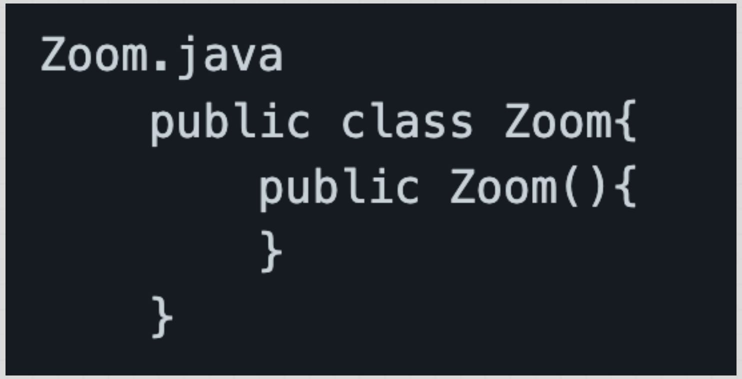
- An object is an instance of a class
- The object has a copy of all the instance variables and instance methods created in the class
- Objects are created by making a constructor class: new ClassName()
- Object types are a datatypes in java. Primitive datatypes are the other.

Instance

- The word instance is used in different contexts
- If you are referring to just an instance of a class that is an object of the class
- Instance members are variables and methods that belong to an object
- Instance is the opposite of static members(which belong to the class)

Constructor

- A constructor is a special method that is invoked when an object is created. It does not have a return type. It is declared with the same name as the class
- It is called in combination with the **new** operator
- Constructors are used to set up the object. In some cases that is initializing the instance variables of the object
- The compiler always created a default constructor if no other constructors were defined in the class
- Constructors can be overloaded. They can be chained. They can also invoke super class constructors (Inheritance)



```
Zoom.java
    public class Zoom{
        public Zoom(){
        }
    }
```

Static

- Static members belong to the class. They are accessed by the class name.
- An object does not need to be created in order to access these member.
- Static variables are useful for accessing data does is fixed. (constants)
- Static methods are useful for utility methods that are meant to be called anywhere quickly

null

- null is a value that acts as a placeholder when no object is assigned to the reference
- It is not an object, just an empty value
- Primitive types cannot be null
- Trying to use a reference that doesn’t have a value yet (object), will cause a NullPointerException

Instantiation

- Instantiation is to create an object
- An object is created from the class by calling the constructor
- When an object is created the memory location of the object is stored into the reference, if there is one.

Constructor Chaining

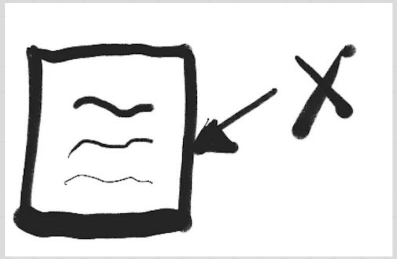
- Constructor chaining is a constructor calling another constructor in the class. This is done with the constructors are overloaded and different parameters
- This idea is used to avoid repetition of the code. Instead of repeating code we already wrote we call a constructor and execute that existing code
- Chaining occurs with this() where the constructor called depends on the given arguments in this()
- The constructor call must be the first line of the constructor

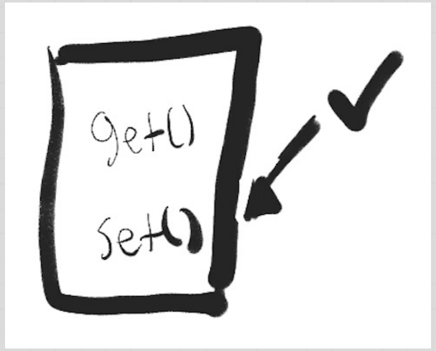
OOP: Object Oriented Programming

- Object Oriented Programming is the approach of how classes, objects and different concepts are working together
- Java is an OOP language
- Objects are instances of a class and how are the classes set up? There is concepts like Inheritance and Polymorphism that explain how all the components we use in java have been working together
- These concepts also define structure for the code, they allow us to create code that is reusable and maintainable

Encapsulation

- Encapsulation protects information by preventing direct access and instead allowing indirect access that is controlled in the way we want the information to be used or read.
- To achieve this data hiding the private access modifier is used on the variables and public getter and setter methods are defined to give the indirect access
- Another way to think of encapsulation is the grouping of information into one unit. That unit is bound together so the information can be accessed through the methods that are defined for it





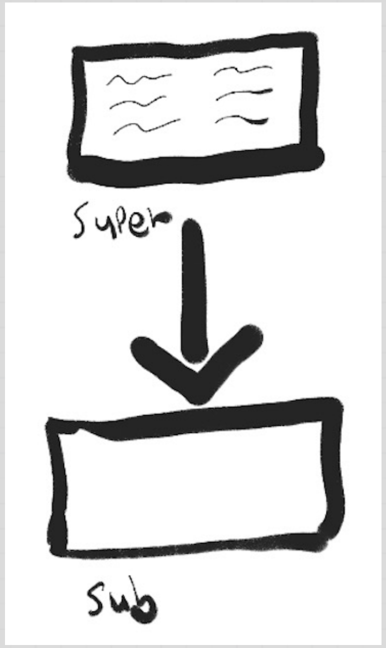
Inheritance

- Inheritance is one of the core concepts that allows java to function. Inheritance allows abstraction and polymorphism to work.

- Inheritance allows one class to pass information and methods to another class. This is important to avoid repetition. This also allows our code to be more organized which results in better maintainability and reusability

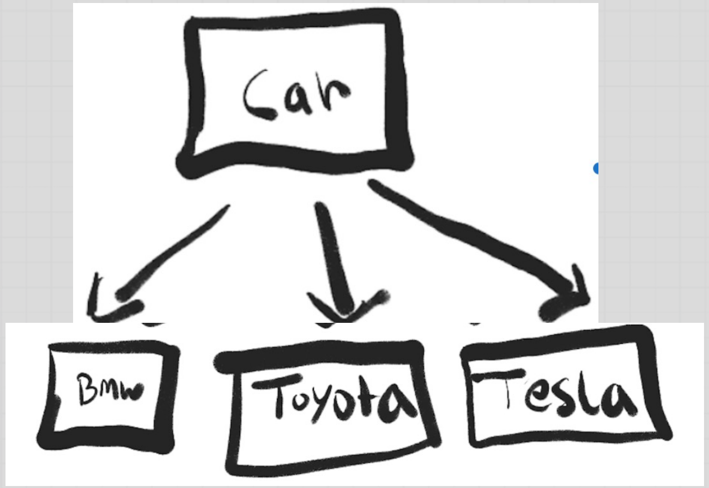
- Super(parent) / sub(child)

- The parent class of all classes is the Object class which gives object types their behavior



Abstraction

- Abstraction is the concept that hides how something done and instead focuses on what is done
- In the case of java, this hiding of how is done through method implementation. Abstraction allows the method to be defined without having any code implementation.
- Abstraction is achieved though abstract class and interfaces is java
- Abstraction allow the structure to be defined early and then details of specifics are added later



Abstract Method

- An abstract method is a method without any implementation. In order words it is a method without any code body
- Abstraction methods can only be contained in an abstract class or interface
- The abstract keyword is added to the method which means the sub classes that inherit this method later must override the method to give implementation to how the method should work in that sub class context

Abstract Class

- An abstract class is like a normal class, but it can create abstract methods
- By making a class abstract the ability to instantiate the class is taken away. Meaning object of the class cannot be created
- The rest of the components of a class are the same: constructors, variables, blocks of code

Interface

- The interface is a not a class, but it can act as a blueprint of a class. They are created in a .java file, but instead of a class an interface is defined

- The main benefit of interface is that they are not classes, which means they don’t need to follow the inheritance rules of a class. In java, a class can only have one direct parent, but interfaces aren’t restricted by that rule. They can inherit and be implemented to as many classes or interfaces as needed

- Many of the common frameworks and libraries follow the interface structure for their abstraction layer because of the flexibility they offer.

  EX: Collection framework in java, Selenium, JDBC

Polymorphism

- Polymorphism allows objects to take different forms, which can happen through the reference

- An object can have 3 different reference types:
  reference of **itself**
  reference of any **super class**
  reference of any **interface** being implemented

- The different references allow an object to be flexible, especially when it comes to methods or data structures

- The interface is a common reference to use since many of abstraction layers define the main functionalities of the objects.
  Ex: Selenium, Collection framework

Method Overriding

- Method overriding allows a methods’ implementation to be changed. More specifically a method that was inherited can be changed.
- One of the benefits of overriding is allowing sub classes to have code that is more relevant rather than having just the inherited parent class implementation. This allows more information or details of the sub classes to be taken into account
- One of the biggest uses of overriding is for implementing abstract methods in sub classes
- Another common overriding use is the toString method() which is called when an object is printed. A sub class is able to define how the object should be printed rather than the default implementation defined in the Object class

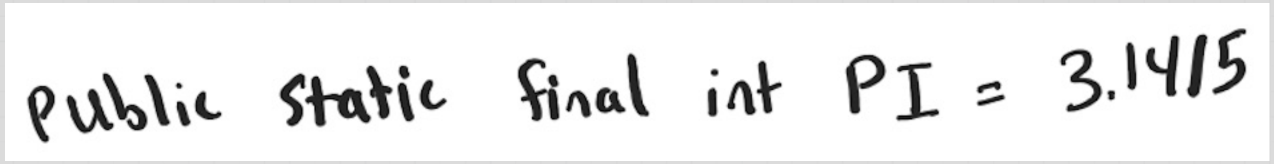
Constant Variable

- Constant variables are for information that is fixed and accessible everywhere

- The syntax for constant variables include **public static final** keywords.
  Public allows access everywhere. Static allows access by the class and no object needed. Final prevents the information from being changed

- Examples of constant in java already Math.PI

- The convention for naming constant variables is all uppercase letters



```
public static final int PI = 3.1415
```

Method Hiding

- Static methods cannot be overridden. Method overriding is meant for instance methods to define a more specific implementation for the sub class
- If you declare a static method in the sub class that has the same name as the the method inherited, you will do method hiding.
- Method hiding is not overriding, but it create another method. Both methods are accessible based on the different reference (super/sub class - where the method was defined). In normal cases a static method that is inherited can be accessed by both references

Error

- An error stops the flow of the program, but an error is not happening because of code. Errors occur based on other factors.
  For example, internet issues, or lack of system resources.

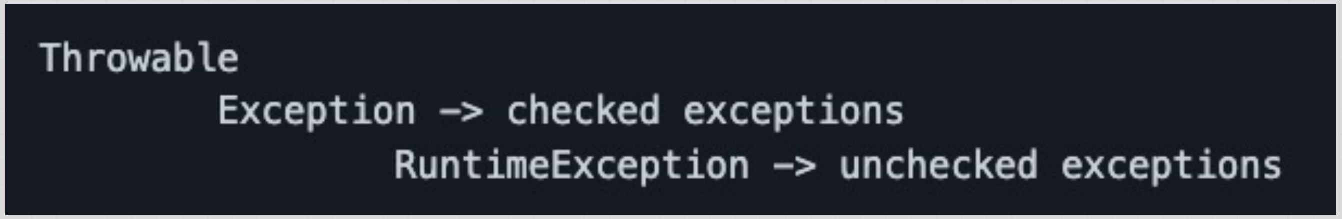
- An error is not meant to be handled by the programmer.

- In terms of inheritance the Errors are sub classes of Throwable

- Common errors: StackOverFlowError, OutOfMemoryError

Exception

- An exceptions stops the normal flow of the program. Exceptions occur because of some code
- A programmer is meant to either handle the exception or fix the code to work properly
- The Inheritance of Exceptions is as follows



```
Throwable
        Exception -> checked exceptions
                RuntimeException -> unchecked exceptions
```

- When an exception is causes the object of the exception class is thrown

Checked Exception

- A checked exception is an exception that must be handled, otherwise the program will not compile. Also called compile time exception.
- Checked exceptions are sub classes of the Exception class
- An example of a method that has a checked exception inside is Thread.sleep(). So, if you want to use this method you must handle the exception

Unchecked Exception

- Unchecked exceptions occur during the execution of the program. Also called run time exception

- Unchecked exceptions are sub classes of the RuntimeException class

- They can be handled but in many cases the code needs to be fixed if the problem was related to a mistake in the code

- Common examples of unchecked exceptions are StringIndexOutOfBounds or NullPointerException

Collection Framework

- The Collection Framework in java is a group of interfaces and classes that define data structures. These data structures differ in their implementations of algorithms for how the data is read, manipulated, or added
- The main interfaces in this framework are Collection, List, Set, and Queue. The Map interface does not inherit the Collection interface, but it is still considered part of the collection framework
- Why have so many collections? The different implementations of the abstract ideas allow data to handled effectively for a specific task
- The framework is a good example of the OOP concepts. Although all these collections exist, ArrayList and HashMap are the most commonly uses collections. It also depends on your practical use cases

Generics

- Generic types provide flexibility with the datatypes allowed when using a specific class
- A class can define generic types and use those types throughout the code in the class. The syntax of them is <> with a letter defining the generic type used in the rest of the code
- A common use case of these can be seen in the Collection classes which can work with whichever object type is provided or it is also used with functional interfaces

List<E>

Map<K,V>

Programming Language

- A programming language is a language that can help us speak with computers. We can write instructions, which is the code, to define the functions we would like the computer to perform.

- Java is a high-level programming language, so we can write human understandable code that will later be translated for the computer to run it

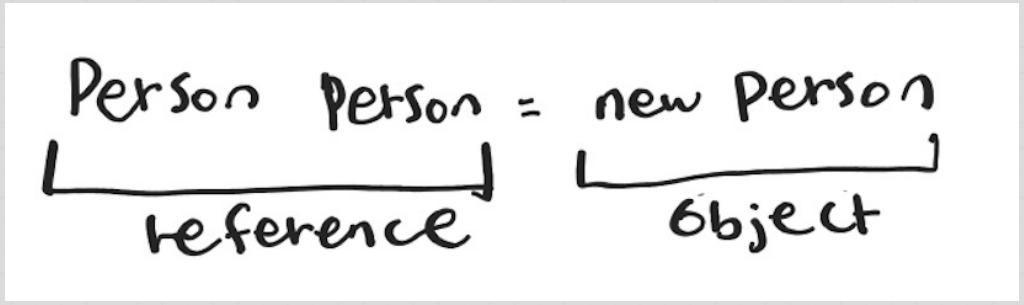
- There is many programming languages as well because different domains will sure different languages that are best suited for their needs

Reference

- The reference is a pointer of an object. Objects are the components that require resources from the computer to represent different data and the reference is the name that gives access to that information

- The reference on its own is just a datatype and name. It needs an object to have any functionality, otherwise if a reference is attempted to be used with no object the NullPointerException is thrown

- The reference is also what gives flexibility to an object through Polymorphism


$$\underbrace{\text{Person person}}_{\text{reference}} = \underbrace{\text{new Person}}_{\text{Object}}$$

Implementation

- Implementation is the exact function/action of a method. There can different implementation to achieve the same result

- Method overriding allows methods to have a different implementation of the code in the sub class

- Abstract methods must also be implemented at some point to complete the idea of abstraction

- Example of implementation:

  expected result: going into your house

  possible implementations to achieve result:

  - go through front door
  - go through back door
  - go through garage
  - go through window

Optimize

- Optimization is always something to think about but not something that needs to be prioritized

- Optimizing the code mean it is working as efficiently as possible. The code may be doing the function you expect, but maybe it is taking too much time or taking too many resource

- For example, if there is a nested loop that will take more execution time than just one loop. Optimizing that loop would be trying to remove the nested loop and being able to do the same functionality with only one loop or maybe even with some ready-made methods

Refactor

- In general, refactoring is updating or changing the code
- Refactoring can be used in different context. The first is when editing the code. Changing the variable name after it's been used multiple time already can be refactoring because the code was changed to improve the variable to be more understandable
- Another situation could be a block of code that is doing some function, but then extracting that code into a method so it can now be reusable instead of only belonging to the one code block

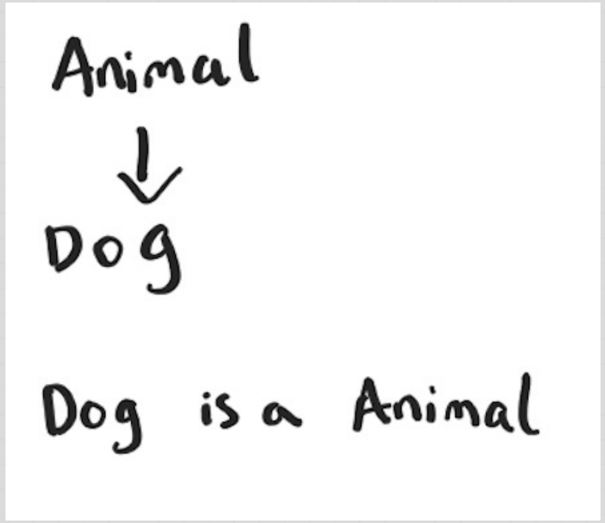
Is A relationship

- Is A relationship describing how two classes and interfaces are related to each other in inheritance

- When a class is inherited, or interface is implemented it creates an Is a relationship that can be explained using this phrasing

  Sub class is a super class
  or
  Sub class is a interface

- Is a relationship determines possible references for an object in Polymorphism



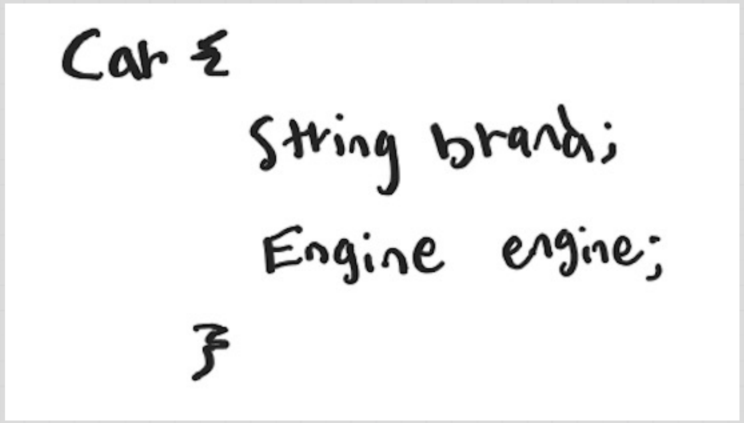
Has A relationship

- Has a relationship can describe when an object has access to another object as an instance member

- For example, if you have a Car class that has an Engine class reference this means the Engine belongs to the Car, so you can say:

    Car has a Engine

- The phrasing helps to describe the association of the classes. How the classes and objects are related to each other



```
Car {
        String brand;
        Engine engine;
    }
```

Car has a Engine