BINUS UNIVERSITY

GREATER JAKARTA • BANDUNG • MALANG

People
Innovation
Excellence

COMP6048001
Data Structures

Introduction to Data Structure and Object-Oriented Programming
Week 1

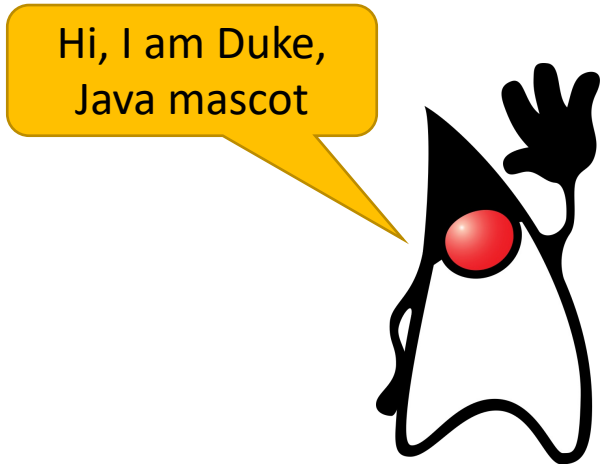
Maria Seraphina Astriani
seraphina@binus.ac.id

Hello World

```
// Waw C++ ^_^

#include <iostream>

int main() {
      std::cout << "Hello World! I am Sera (D3697)";
      std::cout << "seraphina@binus.ac.id";
      return 0;
}
```

Hello World

Java SE: case sensitive




```
// Your First Program

public class HelloWorld {
    public static void main(String[] args) {
        System.out.print("Hello World! I am Sera (D3697)\n");
        System.out.print("seraphina@binus.ac.id");
    }
}
```

There is also `println`

https://1000logos.net/java-logo/


Join Group
WhatsApp

Materials

- Link: https://binusianorg-my.sharepoint.com/personal/seraphina_binus_ac_id/_layouts/15/guestaccess.aspx?guestaccesstoken=8rwAZtqRqnhvVefBQWGkKeSV6PzGBmnTNtraSoWh6rU%3D&folderid=2_1832eff7f98b34d8991fee65fae0dde69&rev=1&e=ocszAl

Materials

- Link: https://binusianorg-my.sharepoint.com/personal/seraphina_binus_ac_id/_layouts/15/guestaccess.aspx?guestaccesstoken=8rwAZtqRqnhvVefBQWGkKeSV6PzGBmnTNtraSoWh6rU%3D&folderid=2_1832eff7f98b34d8991fee65fae0dde69&rev=1&e=ocszAl

- **https://s.id/1z6R4**

BINUS UNIVERSITY

People
Innovation
Excellence

Data Structures???

What does it mean?

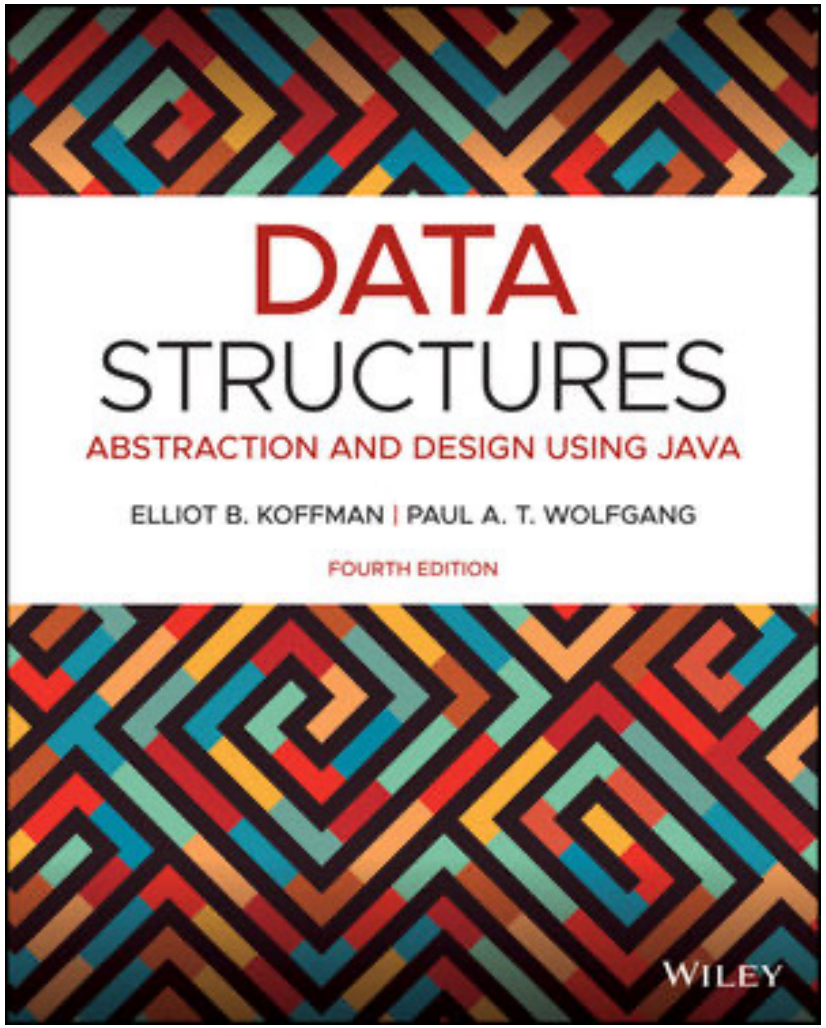
Schedule

Week	Topics	References	Learning Outcomes
1.	• Introduction to Data Structure • Object-Oriented Programming (OOP)	Chapter 1	**LO 1, LO 2**
2.	• Primitive Data Type and Reference Variable	Appendix A.2	**LO 1, LO 2, LO 3, LO 5**
3.	• Algorithm Efficiency • ArrayList	Chapter 2.1 Chapter 2.2, 2.3, 2.4	**LO 1, LO 2, LO 3, LO 5**
4.	• Linked List • Java Collections Framework Design	Chapter 2.5, 2.6, 2.7, 2.8 Chapter 2.10	**LO 1, LO 2, LO 3, LO 5**
5.	• ADT Stacks and Queue	Chapter 4	**LO 2, LO 3, LO 4**
6.	• Recursion	Chapter 5	**LO 2, LO 3, LO 4**
7.	• Fun Games Application • Presentation of Student Project Proposal	[DSPS] Chapter 10	**LO 2, LO 3, LO 4, LO 5, LO 6**
8.	• Trees	Chapter 6	**LO 2, LO 3, LO 4, LO 5**
9.	• Sets and Maps	Chapter 7	**LO 2, LO 3, LO 4**
10.	• Sorting	Chapter 8	**LO 2, LO 3, LO 4, LO5**
11.	• Self Balancing Search Tree	Chapter 9	**LO 3, LO 4, LO 5**
12.	• Graph	Chapter 10	**LO 2, LO 3, LO 4, LO 5**
13.	• **Student Project Presentation**	**N/A**	**LO 2, LO 3, LO 4, LO 5, LO 6**

Learning Outcomes

- LO 1. Describe the use of various data structures
- LO 2. Apply appropriate operations for maintaining common data structures
- LO 3. Apply appropriate data structures and simple algorithms for solving computing problems.
- LO 4. Design computer programs by applying different data structures and related algorithms
- LO 5. Explain the efficiency of some basic algorithms
- LO 6. Design efficient software solutions that are appropriate for specific problems

Text and Other Resources



Text

- *Koffman, E. B., & Wolfgang, P. A. (2021). Data structures: abstraction and design using Java (4th Edition). John Wiley & Sons. [DSA]*

Other Resources

- *Weiss, M. A. (2010). Data Structures and Problem Solving Using Java (Fourth Edition). Addison-Wesley. [DSPS]*
- *Weiss, M. A. (2014). Data structures and algorithm analysis in Java (3rd Ed). Pearson.*
- *Karumanchi, N. (2017). Data Structures and Algorithms Made Easy In JAVA. CareerMonk.*
- *Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data structures and algorithms in Java (6th Ed.). John Wiley & Sons.*

Assessment

No.	Components	Percentage	Learning Outcomes
1.	Quizzes	20 %	LO 1, LO 2, LO 3
2.	Assignments	10 %	LO 1, LO 2
3.	Laboratory	20 %	LO 2, LO 3, LO 4
4.	Project	30 %	LO 2, LO 3, LO 4, LO 6
5.	Final Examination	20 %	LO 1, LO 2, LO 3, LO 5
Total		100 %	

Project

Project presentation + demo + documentation

Programming language? Java SE

Final Project Requirements -> binusmaya, session 1

Group

- Max. 3 students

Q & A

Session Learning Outcomes

Upon successful completion of this course, students are expected to be able to:

- LO 1. Describe the use of various data structures
- LO 2. Apply appropriate operations for maintaining common data structures

Topics

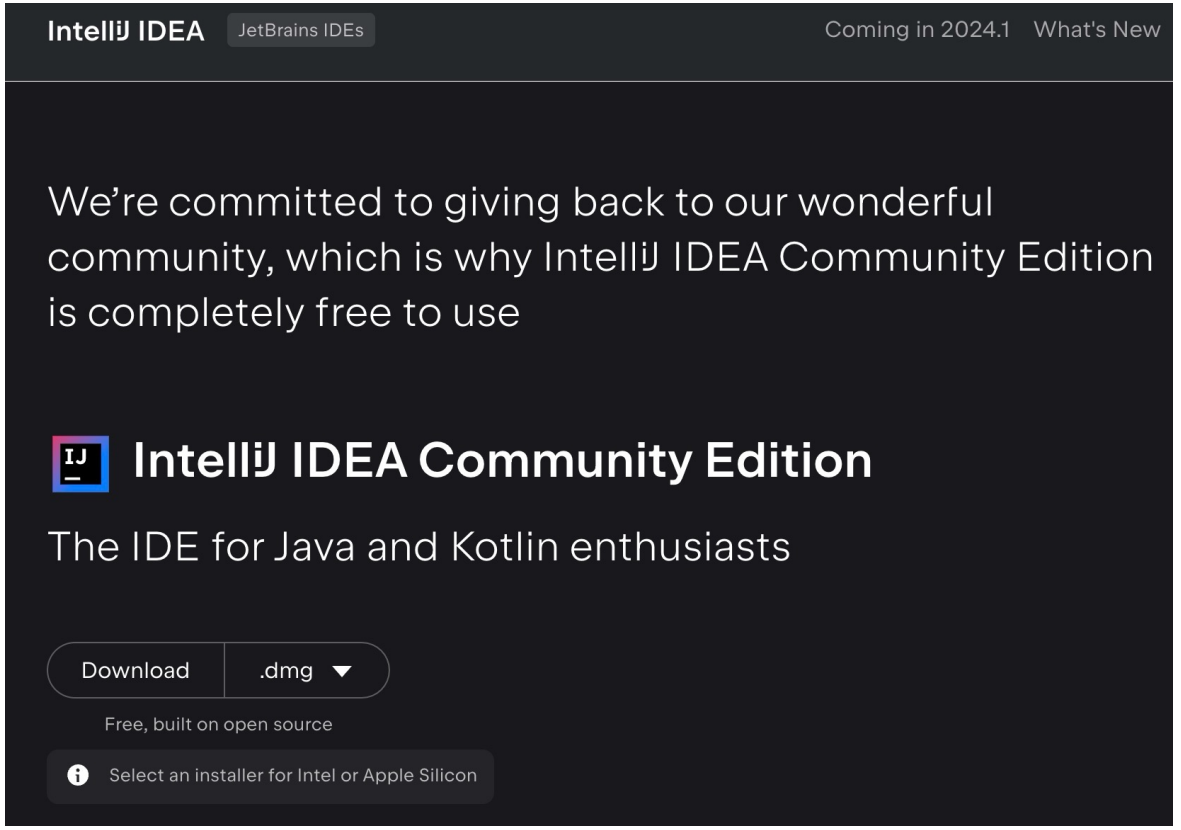
- Introduction to Data Structure
- Object-Oriented Programming

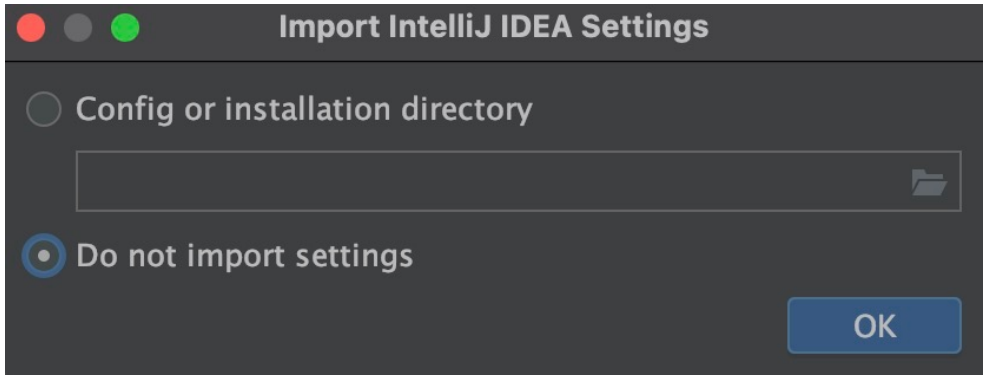
FYI

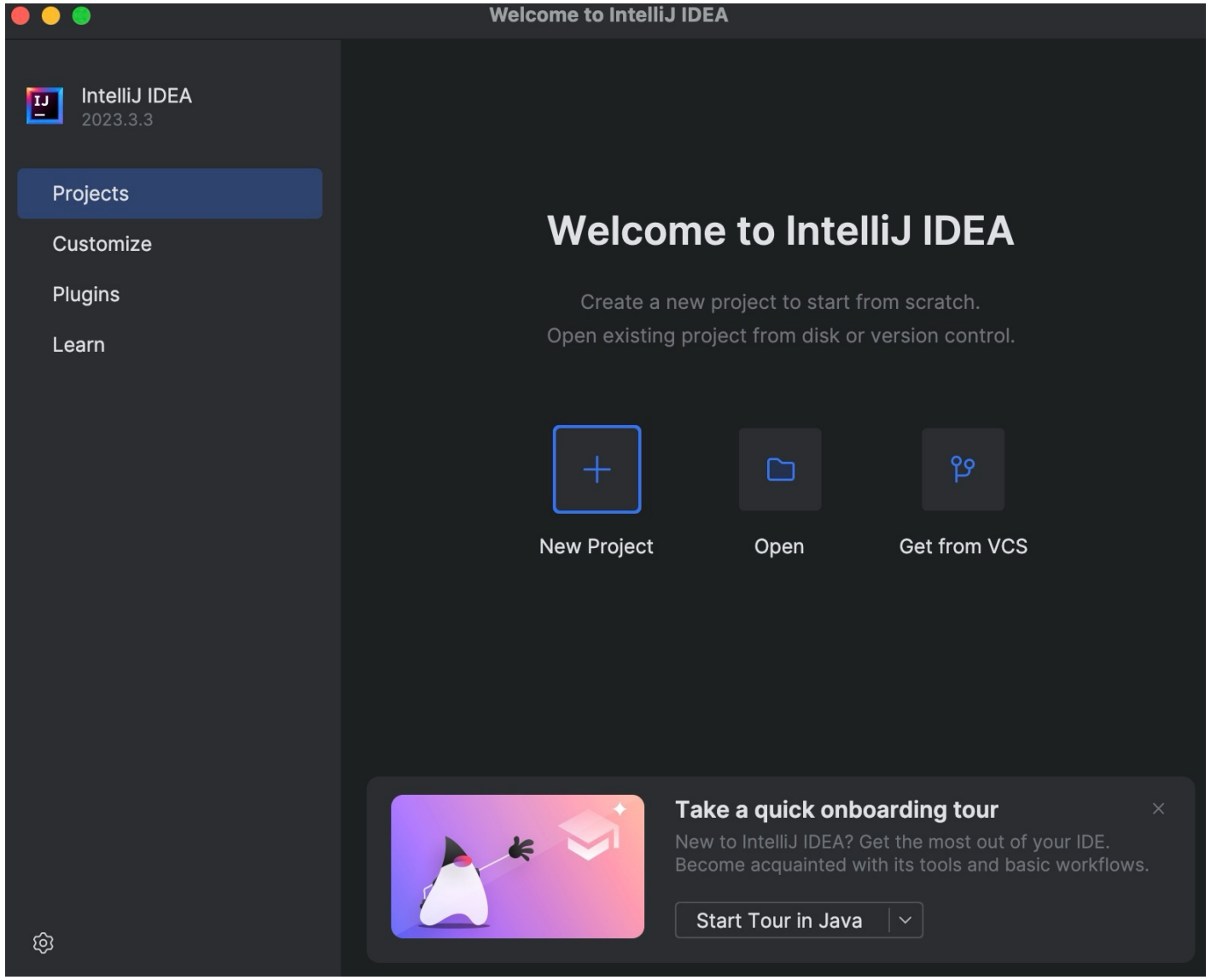
- Java (SE)
  - https://www.oracle.com/java/technologies/java-se-glance.html
  - Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require.
- IDE
  - IntelliJ
  - Eclipse
  - NetBeans
  - etc.

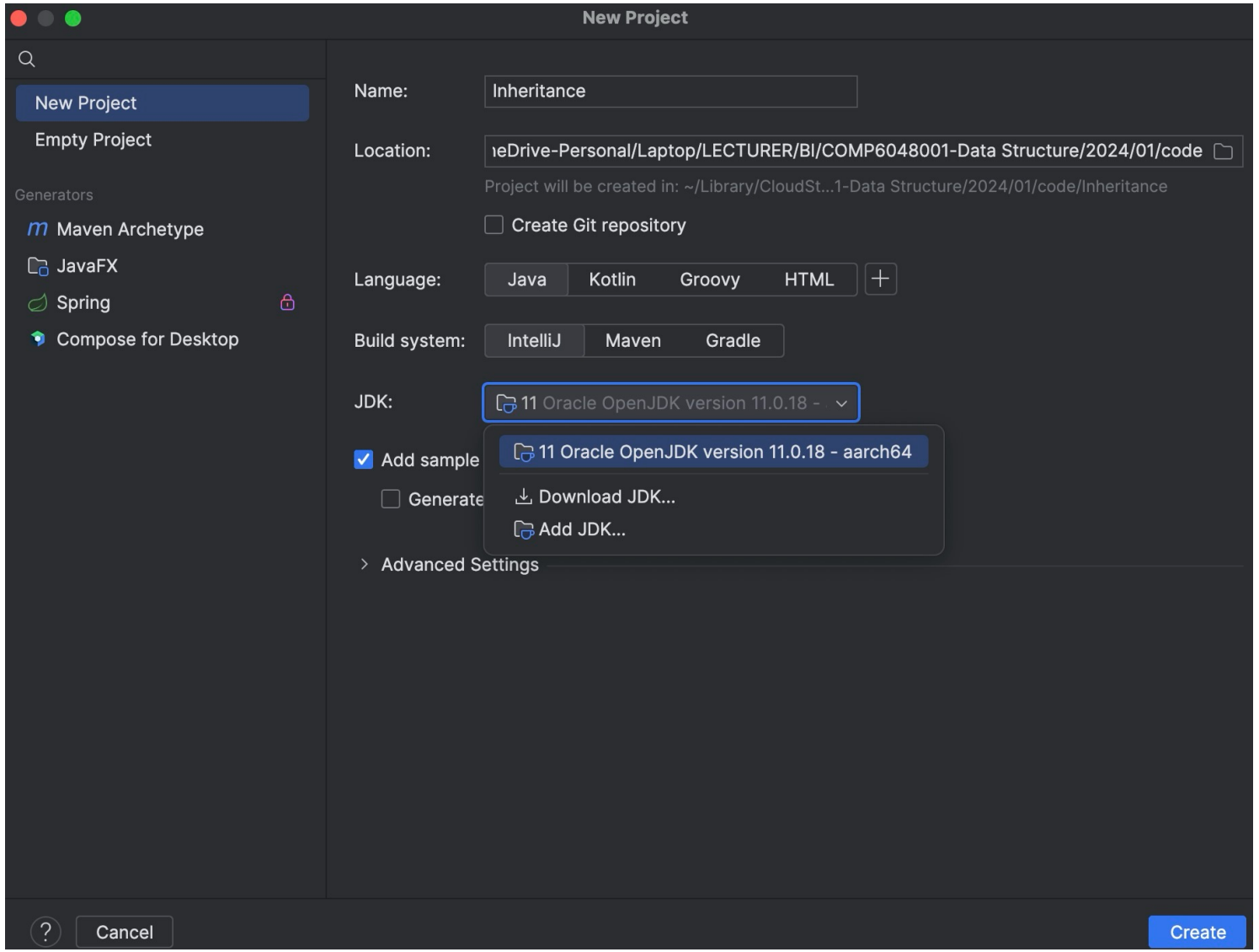
FYI

- Download IDE (https://www.jetbrains.com/idea/download/) and install it. Download the community edition
- Java SE
  - https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html
- Tutorials:
  - https://www.javatpoint.com/java-programs
  - https://www.w3schools.com/java/
- Online compiler:
  - https://www.tutorialspoint.com/online_java_compiler.php
  - https://www.jdoodle.com/online-java-compiler/
  - https://www.programiz.com/java-programming/online-compiler/




Import IntelliJ IDEA Settings
Config or installation directory
Do not import settings
OK


Welcome to IntelliJ IDEA
IntelliJ IDEA
2023.3.3
Projects
Customize
Plugins
Learn
Welcome to IntelliJ IDEA
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
Take a quick onboarding tour
New to IntelliJ IDEA? Get the most out of your IDE.
Become acquainted with its tools and basic workflows.
Start Tour in Java

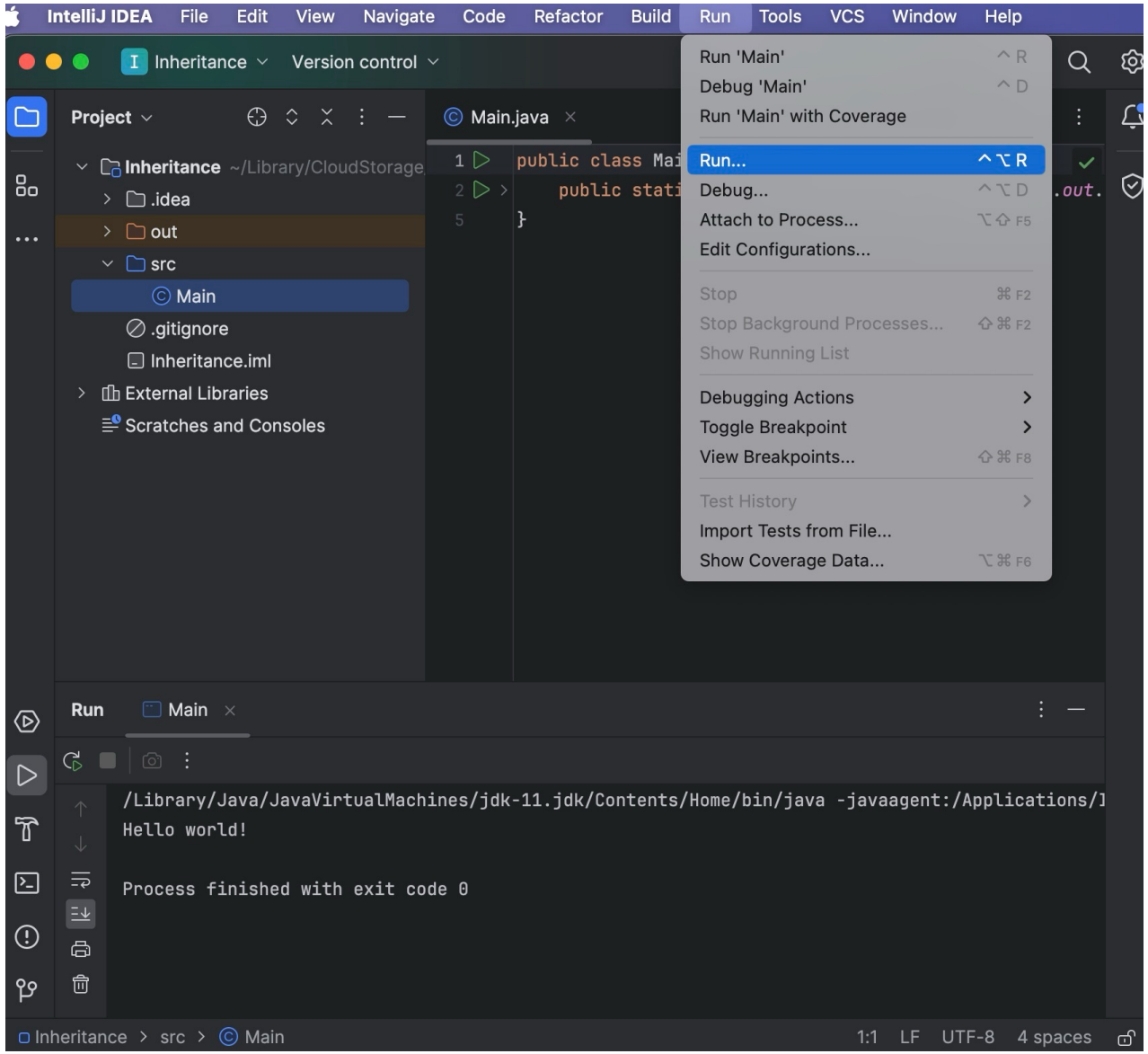


Make Sure You Have The JDK

- Please make sure “New Project” need to be selected
- Named the project
- Select the location of the project (folder)
- Choose: JavaFX
- Language: Java
- Build system: IntelliJ
- Uncheck: Generate …. (below add sample checkbox)

The Java Development Kit (JDK) is a distribution of Java Technology by Oracle Corporation.
In this case, it helps us to compile the code

How To Run It?


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Inheritance Version control
Project
Inheritance ~/Library/CloudStorage
.idea
out
src
Main
.gitignore
Inheritance.iml
External Libraries
Scratches and Consoles
Main.java
public class Mai
public stati
}
Run 'Main' ^R
Debug 'Main' ^D
Run 'Main' with Coverage
Run... ^⌥R
Debug... ^⌥D
Attach to Process... ⌥⇧F5
Edit Configurations...
Stop ⌘F2
Stop Background Processes... ⇧⌘F2
Show Running List
Debugging Actions
Toggle Breakpoint
View Breakpoints... ⇧⌘F8
Test History
Import Tests from File...
Show Coverage Data... ⌥⌘F6
Run Main
/Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home/bin/java -javaagent:/Applications/I
Hello world!
Process finished with exit code 0
Inheritance > src > Main
1:1 LF UTF-8 4 spaces


Main

FYI

- Syntax? For loop, if…else
  - Similar like C, C++, PHP

- Case sensitive

```
int a = 5;

int A = 5;
```

Exercise

- For loop: Print Hello World! 5x

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

- Hint:
  - Declare the variable
  - `for(…;…;…) { … }`

Exercise

- If statement

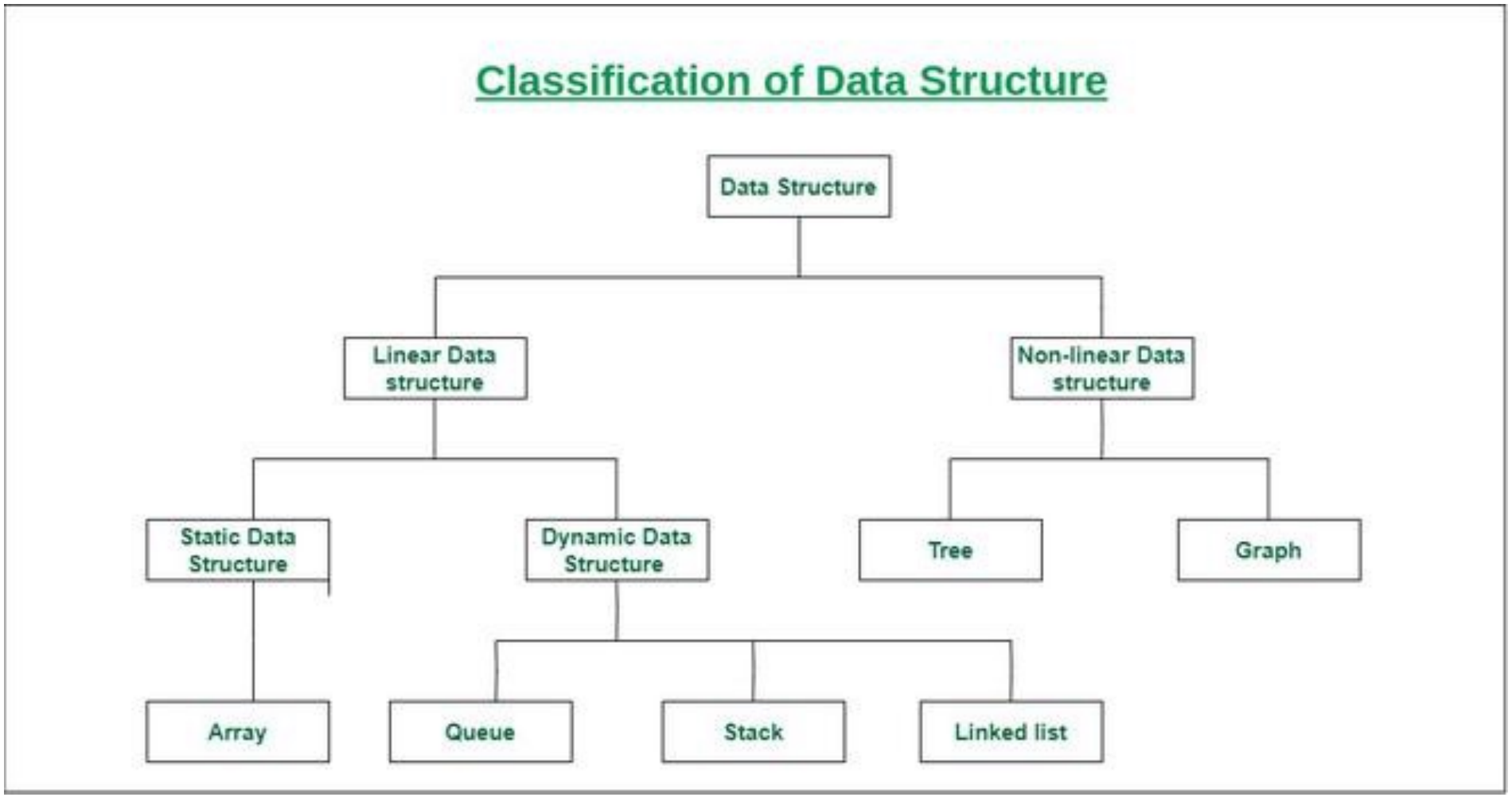
```
1. Hi -Hello world!
2. Hello world!
3. Hi -Hello world!
4. Hello world!
5. Hi -Hello world!
```

Introduction to Data Structure

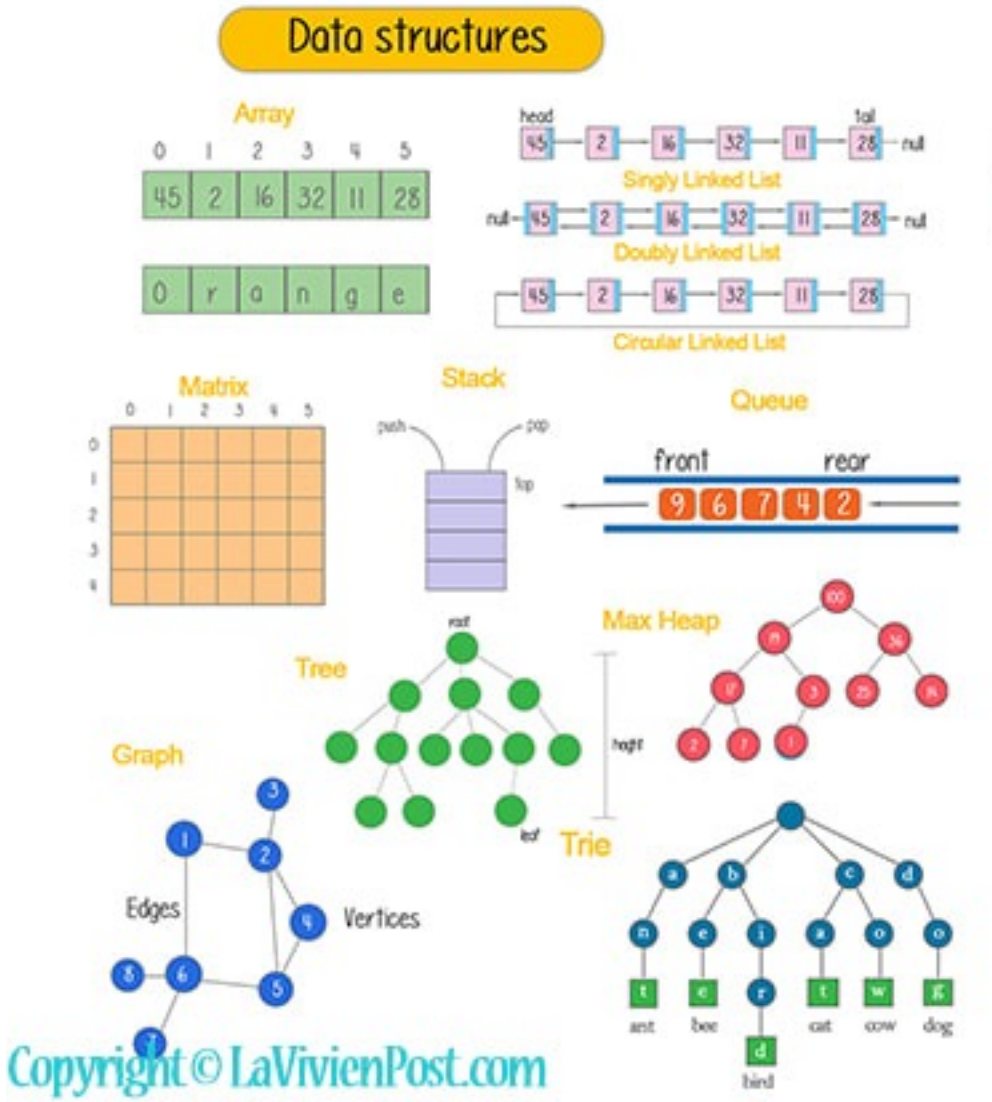
Data Structure

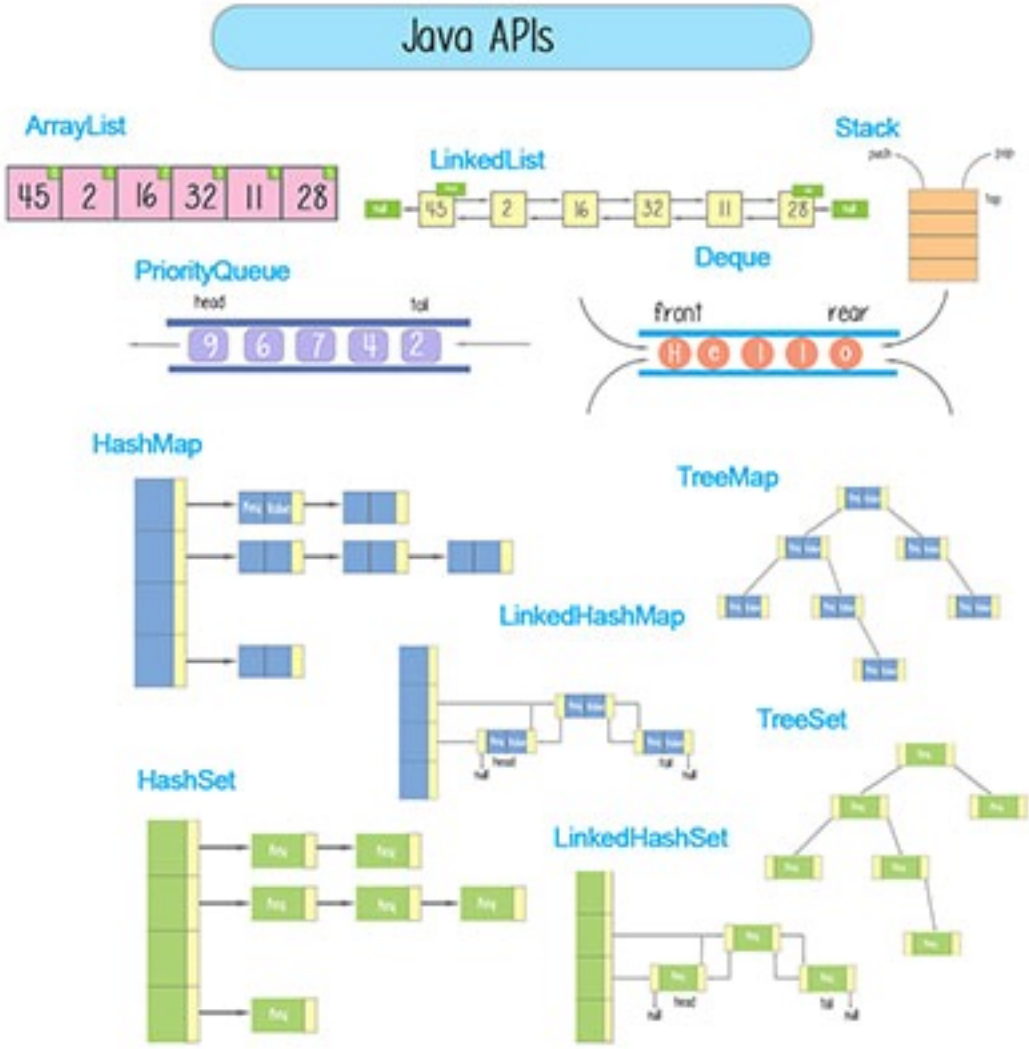
- A data structure is a storage that is used to store and organize data.
- It is a way of arranging data on a computer so that it can be **accessed** and updated **efficiently**.
- When dealing with data structure, we not only focus on one piece of data, but rather different sets of data and how they can relate to one another in an organized manner.

https://www.geeksforgeeks.org/data-structures/
https://www.lavivienpost.com/data-structures-and-java-collections/



https://www.geeksforgeeks.org/data-structures/





https://www.lavivienpost.com/data-structures-and-java-collections/

Java collections are Java built-in library provided by JDK. You don't have to implement your own data structures classes and methods. **You can directly call the library.**

Data structures in Java

- Java data structures example:
  - Arrays
  - Lists
  - Sets
  - Maps

Object-Oriented Programming

Java Application Programming Interface (API)

- Individual class may consist of attributes and methods (operations)
- The use of existing classes (e.g., String and Scanner) may facilitate your programming.
- These classes are part of the Java Application Programming Interface (API).
- Requirement: Java JDK 6 or later (https://docs.oracle.com/outsidein/853/oit/OIPXD/GUID-937039C4-3F99-432F-8DBC-C8F29590C43E.htm#OIPXD2217)

Abstract VS Interface Class

- Abstract class?
- Interface class?

01-Interface vs abstract class.mp4

Abstract VS Interface Class

- Abstract class
  - Contains method(s) and attribute(s)

- Interface class
  - Method(s)
  - interface allows us to implement the same method to classes that have no relationship at all (not in one hierarchy)

Abstract VS Interface Class

- Abstract class:
  - Plane
  - Car
  - Ship

Class:
- Sedan/saloon
- SUV
- MPV
- Etc…

- Interface class:
  - Movable
  - Charging
  - Etc…

How The Abstract Class Will Be?

```
abstract class Bike{
   abstract void run();
}

class Honda4 extends Bike{
      void run(){
            System.out.println("running safely..");
      }

      public static void main(String args[]){
            Bike obj = new Honda4();
            obj.run();
      }
}
```

Abstract Data Type (ADT)

- One way to make code reusable is to encapsulate the data elements together with the methods that operate on that data.
- A new program can then use the methods to manipulate an object's data without being concerned about details of the data representation or the method implementations.
- The encapsulated data together with its methods is called an abstract data type (ADT).

ADT and Abstract Class

- Please refer to https://stackoverflow.com/questions/48839522/is-abstract-class-an-example-of-abstract-data-type

- https://www.geeksforgeeks.org/abstract-data-types/
  - Think of ADT as a black box which hides the inner structure and design of the data type.
  - Example: List ADT, Stack ADT, Queue ADT.
    - To create stack, you may use array or linked-list

Interfaces

- A Java interface is a way to specify or describe an ADT to an applications programmer.
- An interface is like a contract that tells the applications programmer precisely what methods are available and describes the operations they perform.
- It also tells the applications programmer what arguments, if any, must be passed to each method and what result the method will return.

Interfaces

- The interface tells the coder precisely what methods must be written, but it does not provide a detailed algorithm or prescription for how to write them.

- The coder must “program to the interface,” which means he or she must develop the methods described in the interface without variation.

Interfaces - Example

- An automated teller machine (ATM) enables a user to perform certain banking operations from a remote location. It must support the following operations.
- 1. Verify a user's Personal Identification Number (PIN).
- 2. Allow the user to choose a particular account.
- 3. Withdraw a specified amount of money.
- 4. Display the result of an operation.
- 5. Display an account balance.

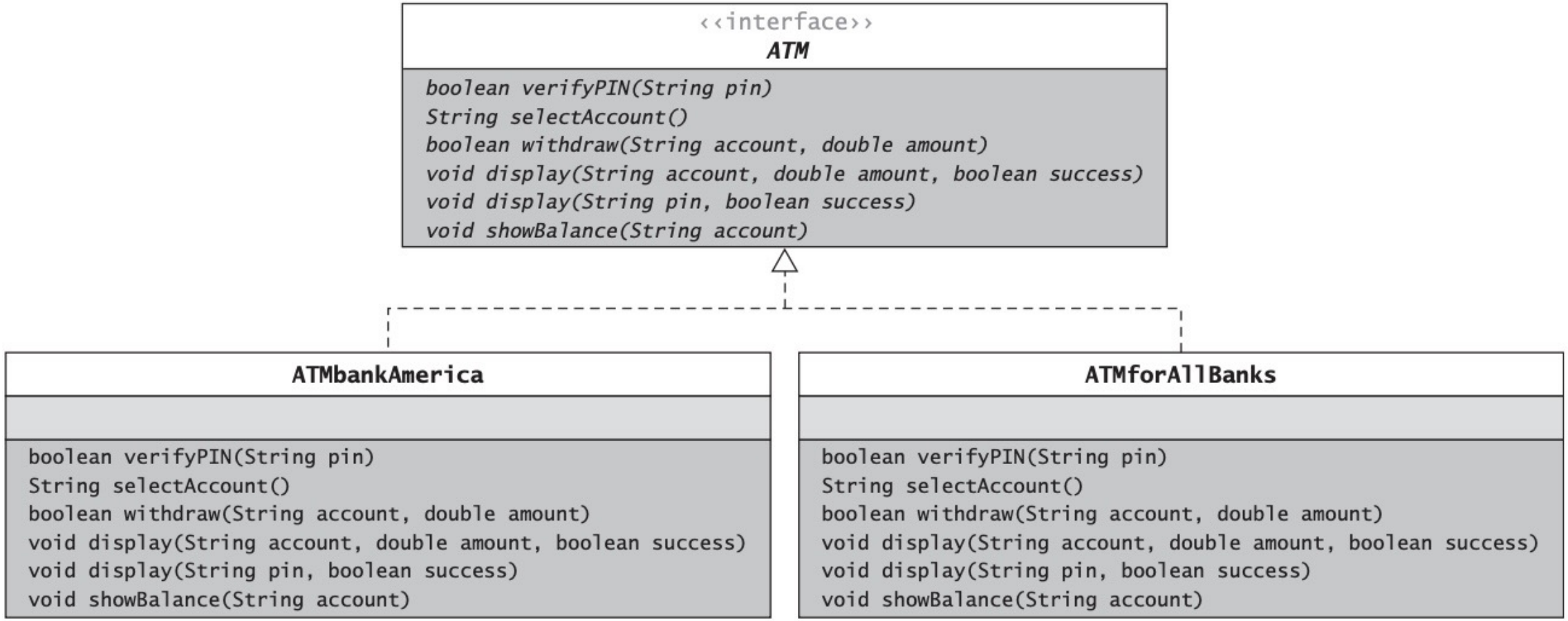
Interfaces - Example

LISTING 1.1
Interface ATM.java

```
/** The interface for an ATM. */
public interface ATM {

    /** Verifies a user's PIN.
        @param pin The user's PIN
        @return Whether or not the User's PIN is verified
     */
    boolean verifyPIN(String pin);

    /** Allows the user to select an account.
        @return a String representing the account selected
     */
    String selectAccount();

    /** Withdraws a specified amount of money
        @param account The account from which the money comes
        @param amount The amount of money withdrawn
        @return Whether or not the operation is successful
     */
    boolean withdraw(String account, double amount);

    /** Displays the result of an operation
        @param account The account for the operation
        @param amount The amount of money
        @param success Whether or not the operation was successful
     */
    void display(String account, double amount, boolean success);

    /** Displays the result of a PIN verification
        @param pin The user's pin
        @param success Whether or not the PIN was valid
     */
    void display(String pin, boolean success);

    /** Displays an account balance
        @param account The account selected
     */
    void showBalance(String account);
}
```

- A class that implements an ATM must provide a method for each operation. We can write this requirement as the interface ATM and save it in file ATM.java.
- The keyword interface on the header line indicates that an interface is being declared.
- The interface definition shows the heading only for several methods. Because only the headings are shown, they are considered *abstract methods.*

UML Diagram Showing the ATM Interface and Its Implementing Classes



Interface

PITFALL

Instantiating an Interface

An interface is not a class, so you cannot instantiate an interface. The statement

```
ATM anATM = new ATM();  // invalid statement
```

will cause the following syntax error:

```
interface ATM is abstract; cannot be instantiated.
```

Object-Oriented Programming (OOP)

- A major reason for the popularity of OOP is that it enables programmers to reuse previously written code saved as classes, reducing the time required to code new applications.

- Because previously written code has already been tested and debugged, the new applications should also be more reliable and therefore easier to test and debug.

Object-Oriented Programming (OOP)

- However, OOP provides additional capabilities beyond the reuse of existing classes.
- If an application needs a new class that is similar to an existing class but not exactly the same, the programmer can create it by extending, or inheriting from, the existing class.
- The new class (called the subclass) can have additional data fields and methods for increased functionality.
- Its objects also inherit the data fields and methods of the original class (called the superclass).

Abstraction, encapsulation, inheritance, and polymorphism are four of the main principles of OOP

FYI – Naming Convention

snake_case

Pros: Concise when it consists of a few words.
Cons: Redundant as hell when it gets longer.
push_something_to_first_queue, pop_what, get_whatever…

skewer-case

Pros: Easy to type.
easier-than-capitals, easier-than-underscore, …
Cons: Any sane language freaks out when you try it.

PascalCase

Pros: Seems neat.
GetItem, SetItem, Convert, ...
Cons: Barely used. (why?)

SCREAMING_SNAKE_CASE

Pros: Can demonstrate your anger with text.
Cons: Makes your eyes deaf.
LOOK_AT_THIS, LOOK_AT_THAT, LOOK_HERE_YOU_MORON, …

camelCase

Pros: Widely used in the programmer community.
Cons: Looks ugly when a few methods are n-worded.
push, reserve, beginBuilding, ...

nocase

Pros: Looks professional.
Cons: Misleading af.
supersexyhippothalamus, **bool** penisbig, …

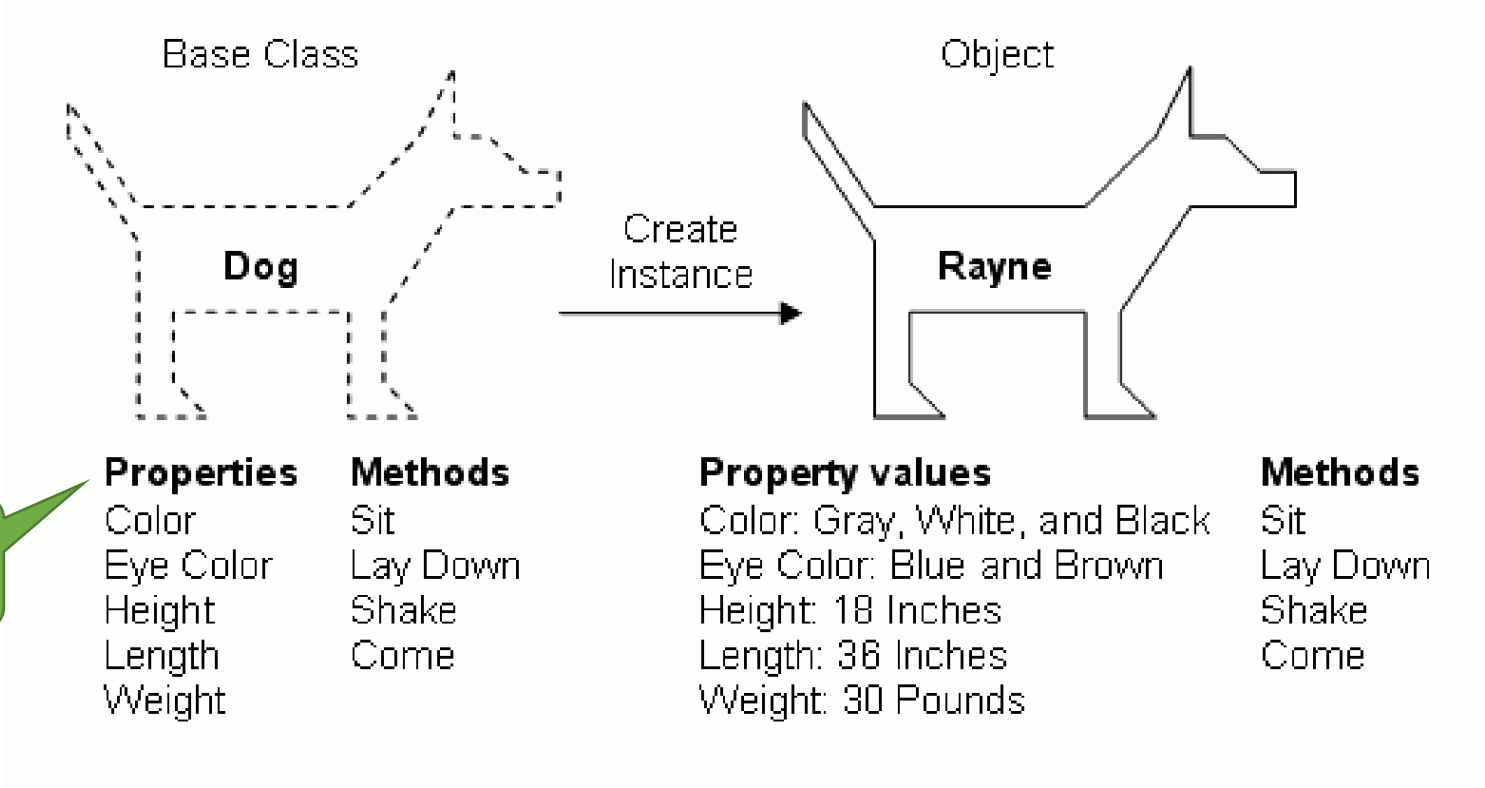
fUcKtHeCaSe

Pros: Can live outside of the law.
Cons: Can be out of a job.

Object-Oriented Programming (OOP)

Class VS Object

- Example:
  - Class: Vegetable
  - Objects: Carrot, Green bean, Spinach.



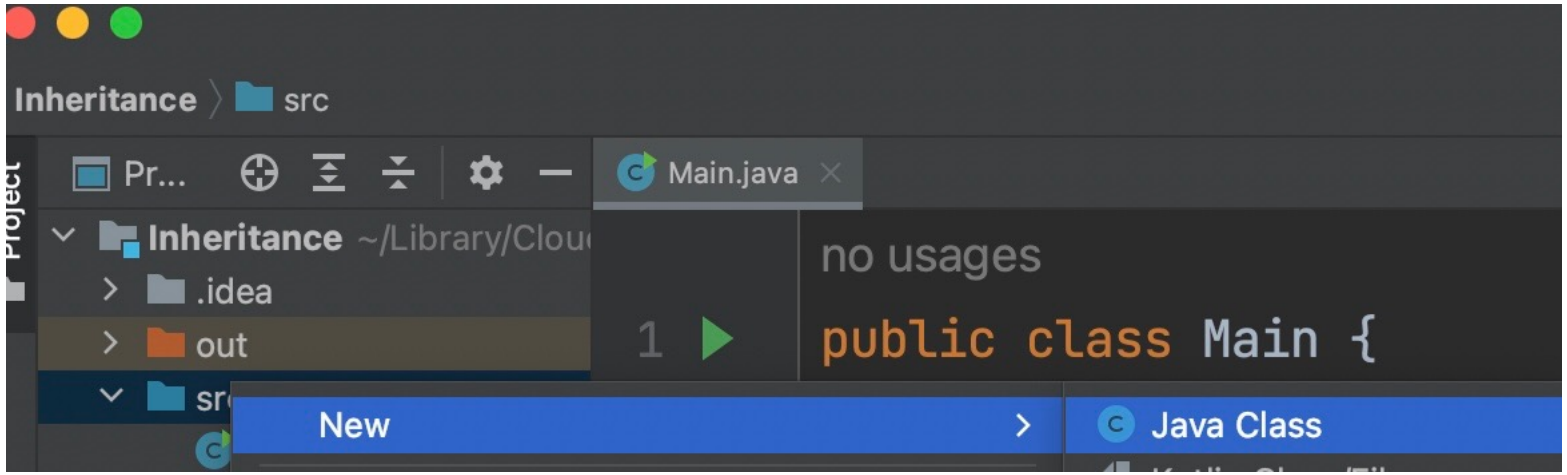
Properties / **attributes**

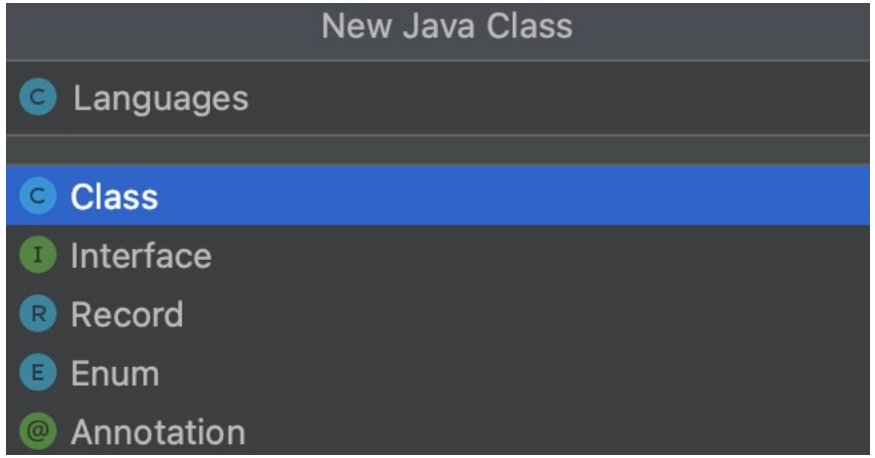
https://www.justsajid.com/skills/csharp/objects-everything-is-an-object-in-c/

Other Example “Dog Barking”

```
Woof woof
Wan wan
Guk guk
```

- Create new class “Languages” (in “src” folder)





Languages.java

```
public class Languages {
    public void dogEnglish()
    {
        System.out.println("Woof woof");
    }

    public void dogIndonesian()
    {
        System.out.println("Guk guk");
    }

    public void dogJapanese()
    {
        System.out.println("Wan wan");
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Languages lang = new Languages();

        lang.dogEnglish();
        lang.dogJapanese();
        lang.dogIndonesian();
    }
}
```

```
Woof woof
Wan wan
Guk guk
```

Let's Enhance It

- Reading input from the keyboard
  - You need to have `import java.util.Scanner;`

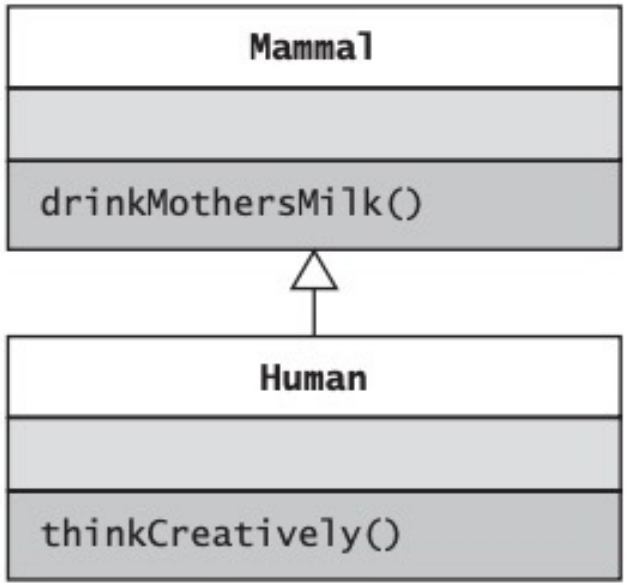
- Pass it to `Languages` class

```
Type something and press Enter: hello world
Woof woof hello world
Wan wan hello world
Guk guk hello world
```

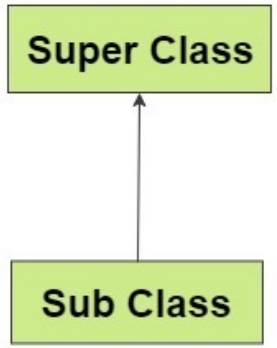
Object-Oriented Programming (OOP)

- Inheritance and hierarchical organization allow you to capture the idea that one thing may be a refinement or an extension of another.
- For example, an object that is a Human is a Mammal (the superclass of Human).
- This means that an object of type Human has all the data fields and methods defined by class `Mammal` (e.g., method `drinkMothersMilk`), but it may also have more data fields and methods that are not contained in class `Mammal` (e.g., method `thinkCreatively`).

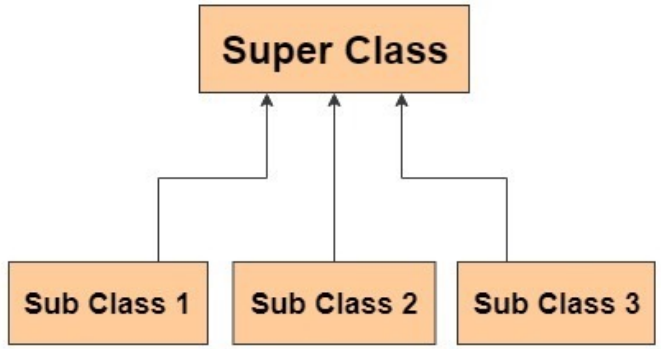
Classes **Mammal** and **Human**



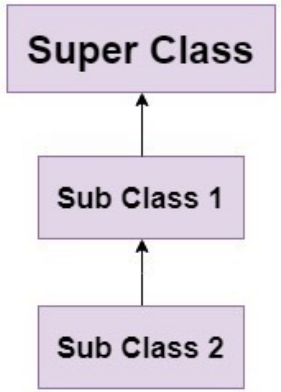
Single Inheritance



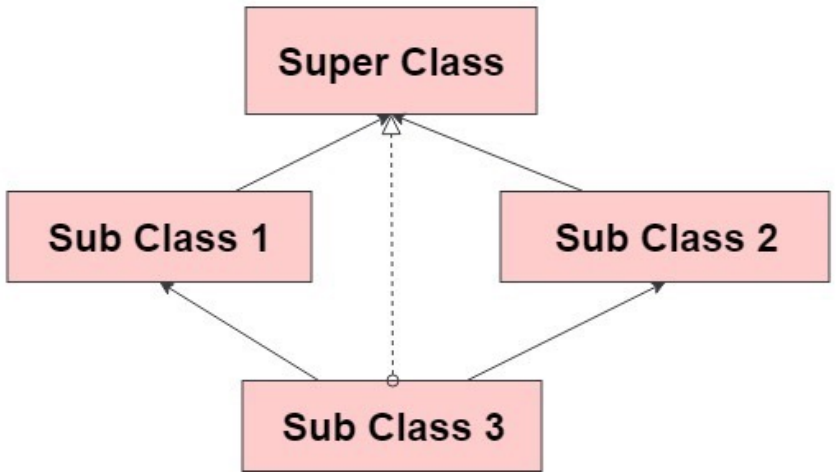
Hierarchial Inheritance



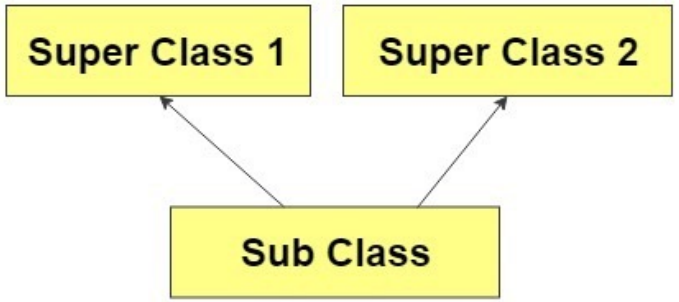
MultiLevel Inheritance



Hybrid Inheritance

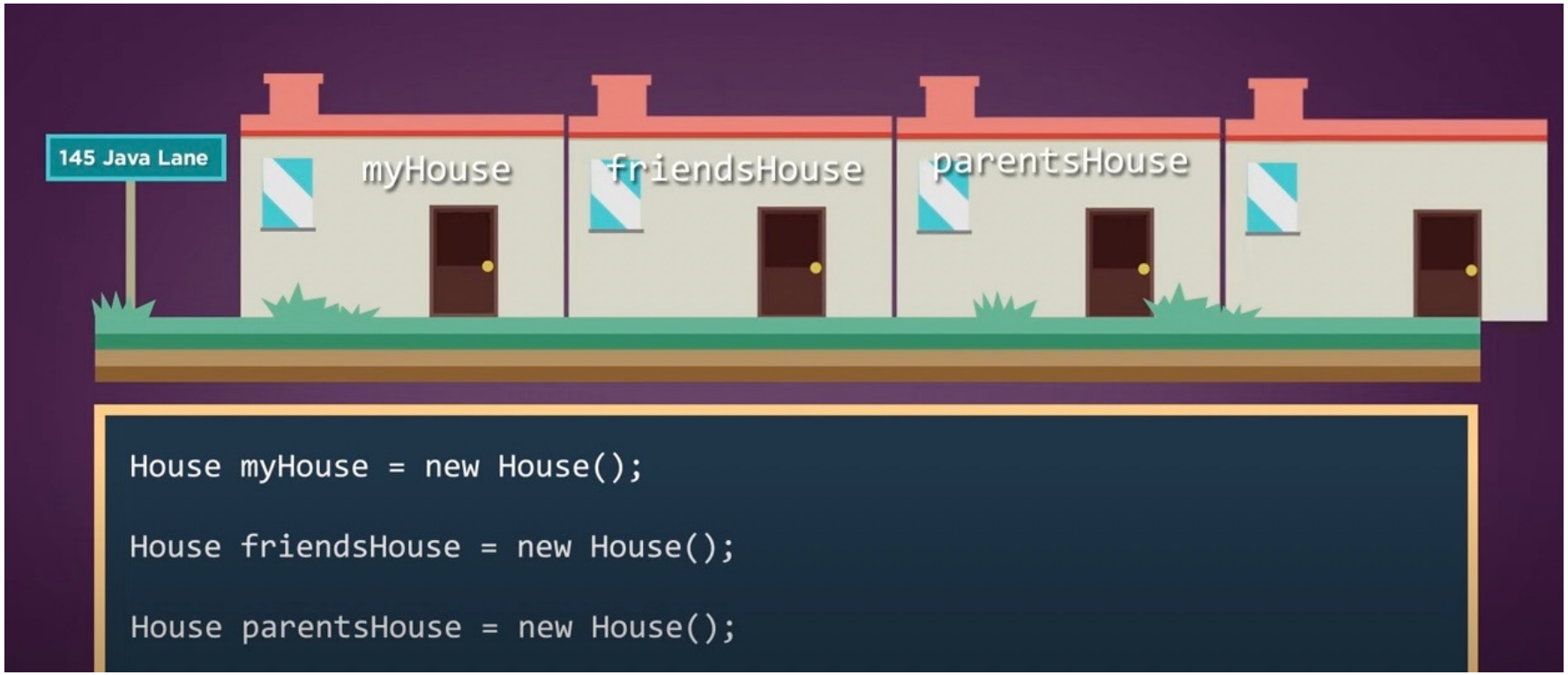


Multiple Inhertance



https://simplesnippets.tech/inheritance-in-java-types-of-inheritance/

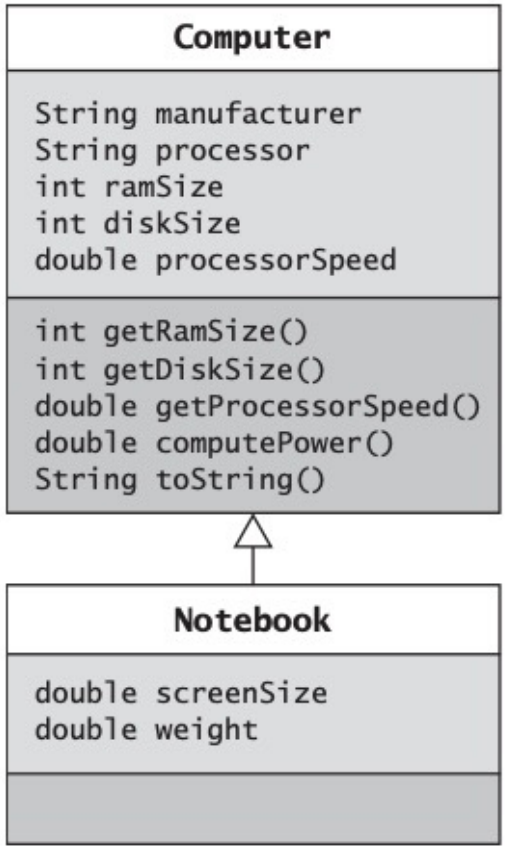
How To Write/Create The Objects?



- 03-CSA_ Creating Objects.mp4

Object-Oriented Programming (OOP)

Classes **NoteBook** and **Computer**



- To illustrate the concepts of inheritance and class hierarchies, let's consider a simple case of two classes: `Computer` and `Notebook`.
- A Computer object has a `manufacturer`, `processor`, `RAM`, and `disk`.
- A notebook computer is a kind of computer, so it has all the properties of a computer plus some additional features (screen size and weight).
- There may be other subclasses, such as tablet computer or game computer, but we will ignore them for now.
- We can define class `Notebook` as a subclass of class `Computer`.

Object-Oriented Programming (OOP)

Class Computer.java

```
/** Class that represents a computer. */
public class Computer {
    // Data Fields
    private String manufacturer;
    private String processor;
    private double ramSize;
    private int diskSize;
    private double processorSpeed;

    // Methods
    /** Initializes a Computer object with all properties specified.
        @param man The computer manufacturer
        @param processor The processor type
        @param ram The RAM size
        @param disk The disk size
        @param procSpeed The processor speed
     */
    public Computer(String man, String processor, double ram,
                    int disk, double procSpeed) {
        manufacturer = man;
        this.processor = processor;
        ramSize = ram;
        diskSize = disk;
        processorSpeed = procSpeed;
    }

    public double computePower() { return ramSize * processorSpeed; }
    public double getRamSize() { return ramSize; }
    public double getProcessorSpeed() { return processorSpeed; }
    public int getDiskSize() { return diskSize; }
    // Insert other accessor and modifier methods here.

    public String toString() {
        String result = "Manufacturer: " + manufacturer +
                        "\nCPU: " + processor +
                        "\nRAM: " + ramSize + " gigabytes" +
                        "\nDisk: " + diskSize + " gigabytes" +
                        "\nProcessor speed: " + processorSpeed + " gigahertz";
        return result;
    }
}
```

Object-Oriented Programming (OOP)

Class **Notebook**

```
/** Class that represents a notebook computer. */
public class Notebook extends Computer {
    // Data Fields
    private double screenSize;
    private double weight;

    // Methods
    /** Initializes a Notebook object with all properties specified.
        @param man The computer manufacturer
        @param proc The processor type
        @param ram The RAM size
        @param disk The disk size
        @param procSpeed The processor speed
        @param screen The screen size
        @param wei The weight
     */
    public Notebook(String man, String proc, double ram, int disk,
                    double procSpeed, double screen, double wei) {
        super(man, proc, ram, disk, procSpeed);
        screenSize = screen;
        weight = wei;
    }
}
```

Let's Code Simple Inheritance Example

```
Woof woof
Dog - Type : land
Legs : 4
```

- Please a new class: InheritanceExample
- Superclass: Animal
- Subclass: Dog

Method Overriding

- Overriding = override the inherited method

```
/** Tests classes Computer and Notebook. Creates an object of each and
    displays them.
    @param args[] No control parameters
 */
public static void main(String[] args) {
    Computer myComputer =
        new Computer("Acme", "Intel", 4, 750, 3.5);
    Notebook yourComputer =
        new Notebook("DellGate", "AMD", 4, 500,
                     2.4, 15.0, 7.5);
    System.out.println("My computer is:\n" + myComputer.toString());
    System.out.println("\nYour computer is:\n" +
                       yourComputer.toString());
}
```

- In the second call to println, the method call

  `yourComputer.toString()`

- applies method toString to object yourComputer (type Notebook).

Method Overriding

- Because class Notebook doesn't define its own toString method, class Notebook inherits the toString method defined in class Computer.
- Executing this method displays the following output lines:

```
My computer is:
Manufacturer: Acme
CPU: Intel
RAM: 4.0 gigabytes
Disk: 750 gigabytes
Speed: 3.5 gigahertz

Your computer is:
Manufacturer: DellGate
CPU: AMD
RAM: 4.0 gigabytes
Disk: 500 gigabytes
Speed: 2.4 gigahertz
```

Method Overriding

- Unfortunately, this output doesn't show the complete state of object yourComputer.
- To show the complete state of a notebook computer, we need to define a toString method for class Notebook.
- If class Notebook has its own toString method, it will override the inherited method and will be invoked by the method call `yourComputer.toString()`. We define method toString for class Notebook next.

```
public String toString() {
    String result = super.toString() +
                    "\nScreen size: " + screenSize + " inches" +
                    "\nWeight: " + weight + " pounds";
    return result;
}
```

Method Overriding

- This method `Notebook.toString` returns a string representation of the state of a Notebook object. The first line

```
String result = super.toString()
```

- uses method call **super**.toString() to invoke the toString method of the superclass (method `Computer.toString`) to get the string representation of the four data fields that are inherited from the superclass. The next two lines append the data fields defined in class Notebook to this string.

Method Overloading

- Having multiple methods with the same name but different signatures in a class is called method *overloading*.

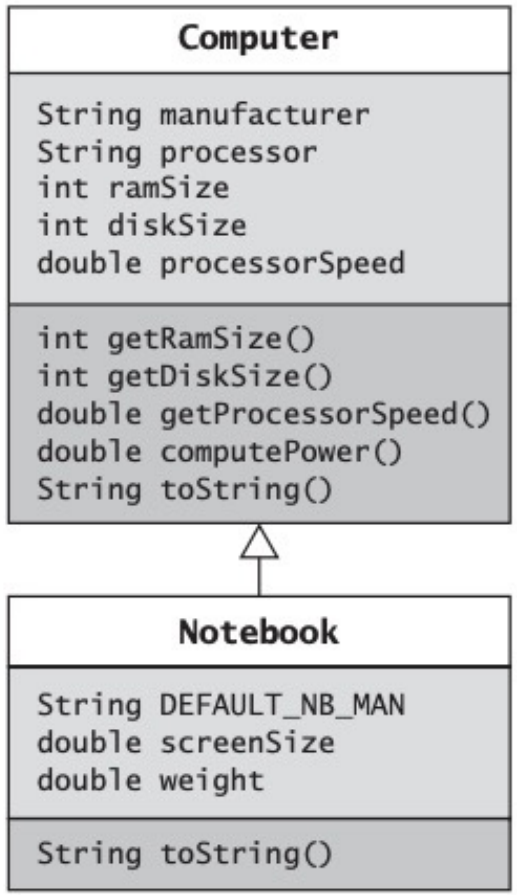
Without Method Overloading

```
int add2(int x, int y)
{
   return(x+y);
}
int add3(int x, int y,int z)
{
   return(x+y+z);
}
int add4(int w, int x,int y, int z)
{
   return(w+x+y+z);
}
```

With Method Overloading

```
int add(int x, int y)
{
   return(x+y);
}
int add(int x, int y,int z)
{
   return(x+y+z);
}
int add(int w, int x,int y, int z)
{
   return(w+x+y+z);
}
```

https://simplesnippets.tech/method-overloading-in-java-explained/

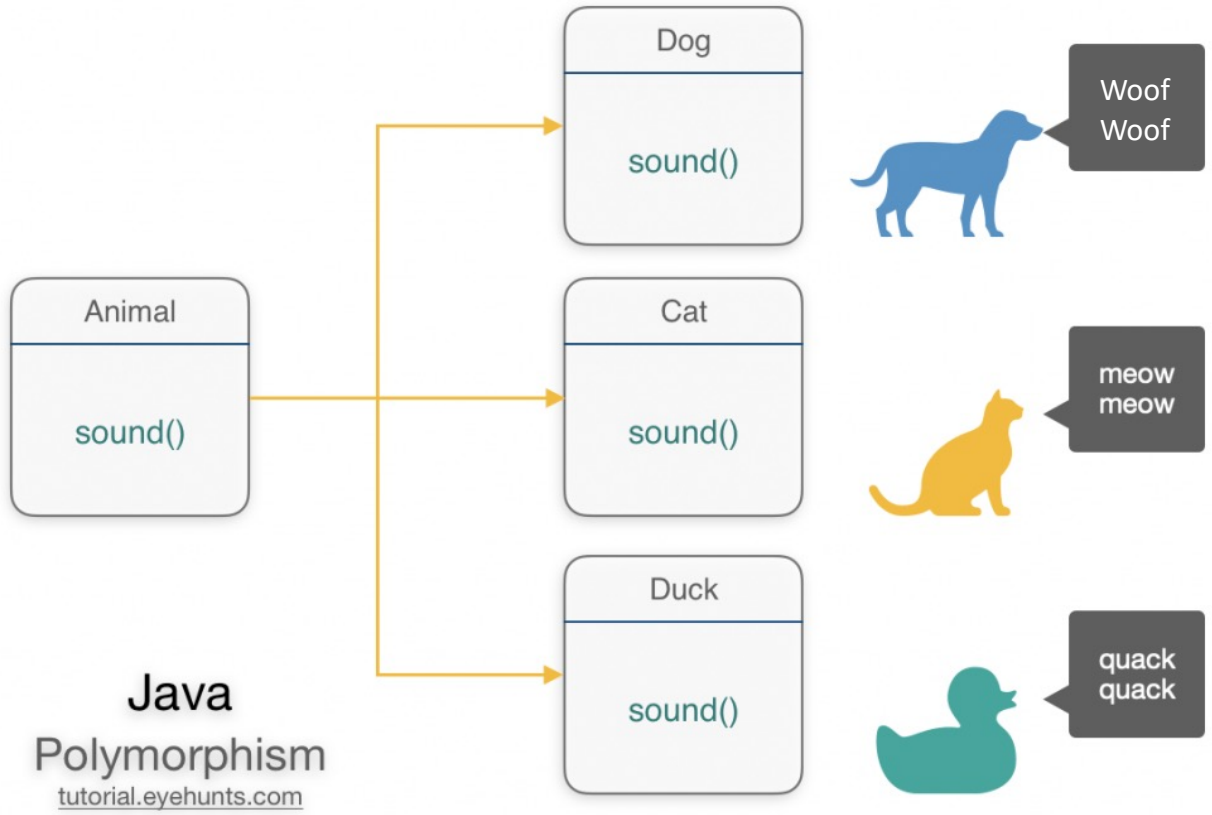



```java
/** Class that represents a notebook computer. */
public class Notebook extends Computer {
    // Data Fields
    private static final String DEFAULT_NB_MAN = "MyBrand";
    private double screenSize;
    private double weight;

    /** Initializes a Notebook object with all properties specified.
        @param man The computer manufacturer
        @param proc The processor type
        @param ram The RAM size
        @param disk The disk size
        @param screen The screen size
        @param wei The weight
     */
    public Notebook(String man, String proc, int ram, int disk,
                    double procSpeed, double screen, double wei) {
        super(man, proc, ram, disk, procSpeed);
        screenSize = screen;
        weight = wei;
    }

    /** Initializes a Notebook object with 6 properties specified. */
    public Notebook(String proc, int ram, int disk,
                    double procSpeed, double screen, double wei) {
        this(DEFAULT_NB_MAN, proc, ram, disk, procSpeed, screen, wei);
    }

    @Override
    public String toString() {
        String result = super.toString() +
                        "\nScreen size: " + screenSize + " inches" +
                        "\nWeight: " + weight + " pounds";
        return result;
    }
}
```

Polymorphism

- An important advantage of OOP is that it supports a feature called *polymorphism*, which means many forms or many shapes.

- Polymorphism enables the JVM to determine at run time which of the classes in a hierarchy is referenced by a superclass variable or parameter.



FYI

- Polymorphism in Java has two types: Runtime polymorphism (dynamic binding) and Compile time polymorphism (static binding).
- Method **overriding** is an example of dynamic polymorphism
- Method **overloading** is an example of static polymorphism.

https://www.sitepoint.com/quick-guide-to-polymorphism-in-java/#:~:text=Polymorphism%20in%20Java%20has%20two,an%20example%20of%20static%20polymorphism.

Polymorphism in Action

- Continue from slide page 56

```
Woof woof
Dog - Type : land
Legs : 4

Cluck cluck
Chicken Type : land
Legs : 2

Psst... no sound
Ant Type : land
Legs : 6
```

Polymorphism

- Suppose you are not sure whether a computer referenced in a program will be a notebook or a regular computer. If you declare the reference variable

```
Computer theComputer;
```

you can use it to reference an object of either type because a type Notebook object can be referenced by a type Computer variable.

Polymorphism

- In Java, a variable of a superclass type (general) can reference an object of a subclass type (specific).
- Notebook objects are Computer objects with more features. When the following statements are executed,

```
theComputer = new Computer("Acme", "Intel", 2, 160, 2.6);
System.out.println(theComputer.toString());
```

you would see four output lines, representing the state of the object referenced by theComputer.

Polymorphism

- Now suppose you have purchased a notebook computer instead.

- What happens when the following statements are executed?

```
theComputer = new Notebook("Bravo", "Intel", 4, 240, 2.4. 15.0, 7.5);
System.out.println(theComputer.toString());
```

Polymorphism

- Recall that `theComputer` is type `Computer`.
- Will the `theComputer.toString()` method call return a string with all seven data fields or just the five data fields defined for a Computer object?
- The answer is a string with all seven data fields.
- The reason is that the type of the object receiving the toString message determines which `toString` method is called.
- Even though variable `theComputer` is type Computer, it references a type Notebook object, and the Notebook object receives the `toString` message.
- Therefore, the method `toString` for class `Notebook` is the one called.

Abstract Classes (Getting Deeper)

- An abstract class is denoted by the use of the word abstract in its heading:

visibility `abstract class` *className*

- An abstract class differs from an actual class (sometimes called a concrete class) in two respects:
  - An abstract class cannot be instantiated.
  - An abstract class may declare abstract methods.

Example of Abstract class that has an abstract method

- In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
abstract class Bike{
   abstract void run();
}

class Honda4 extends Bike{
      void run(){
            System.out.println("running safely..");
      }

      public static void main(String args[]){
            Bike obj = new Honda4();
            obj.run();
      }
}
```

https://www.javatpoint.com/abstract-class-in-java

Class Object and Casting

Class Object

- The class `Object` is a special class in Java because it is the root of the class hierarchy, and every class has Object as a superclass.

- All classes inherit the methods defined in class `Object`; however, these methods may be overridden in the current class or in a superclass (if any)

Class Object and Casting

- The Class `Object`

Method	Behavior
`boolean equals(Object obj)`	Compares this object to its argument
`int hashCode()`	Returns an integer hash code value for this object
`String toString()`	Returns a string that textually represents the object
`Class<?> getClass()`	Returns a unique object that identifies the class of this object

Class Object and Casting

Casting

- Java provides a mechanism, *casting*, that enables us to process the object referenced by `aThing` through a reference variable of its actual type, instead of through a type Object reference.
- The expression

```
(Integer) aThing
```

  casts the type of the object referenced by `aThing (type Object)` to type Integer.
- The casting operation will succeed only if the object referenced by aThing is, in fact, type Integer; if not, a `ClassCastException` will be thrown.

Inheritance Example – The Exception Class Hierarchy

- Java uses inheritance to build a class hierarchy that is fundamental to detecting and correcting errors during program execution (run-time errors).
- A run-time error occurs during program execution when the Java Virtual Machine (JVM) detects an operation that it knows to be incorrect.
- A run-time error will cause the JVM to throw an exception— that is, to create an object of an exception type that identifies the kind of incorrect operation and also interrupts normal processing

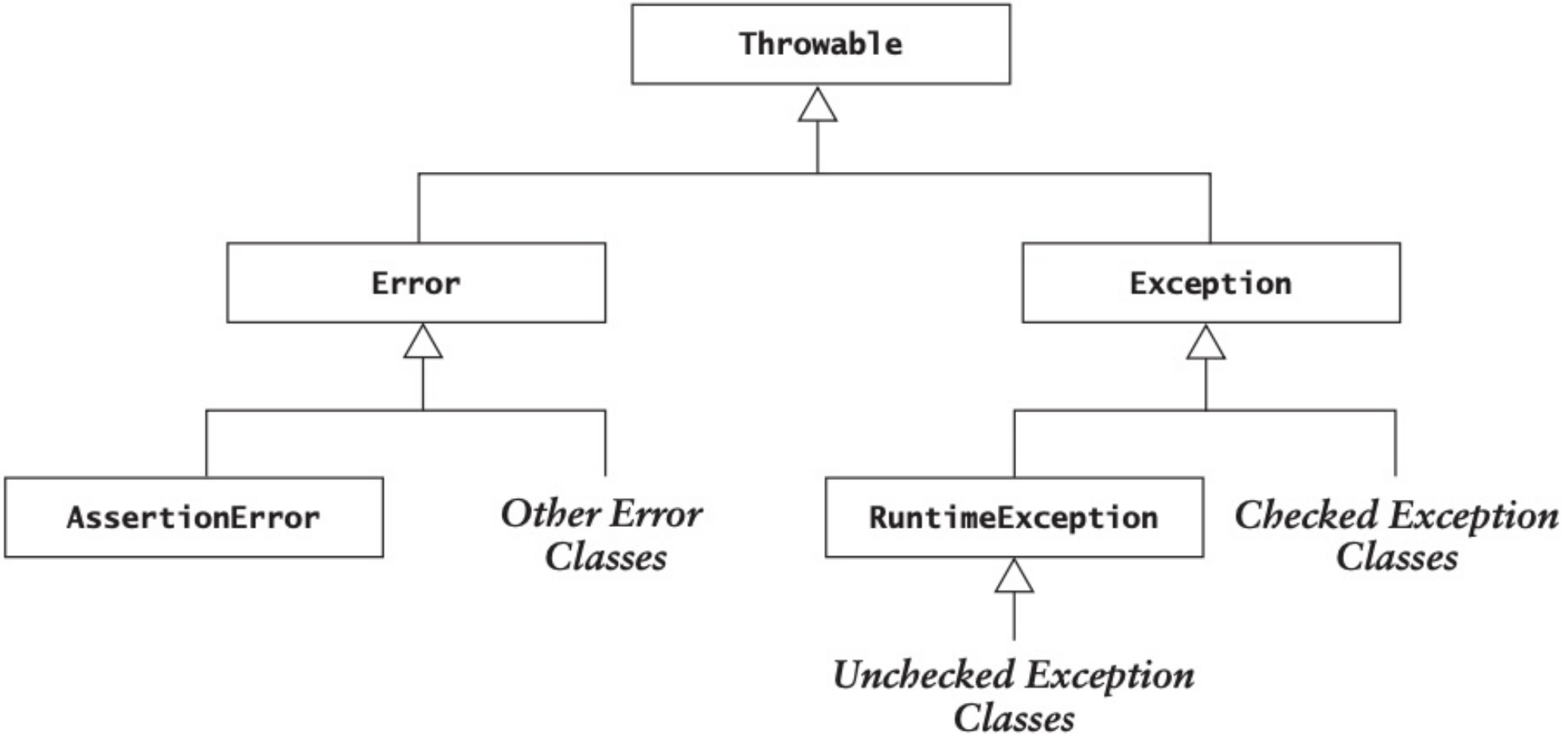
Inheritance Example – The Exception Class Hierarchy

- Some examples of exceptions that are run-time errors.
- All are subclasses of class `RuntimeException`. Following are some examples of the exceptions listed in the table.

Subclasses of java.lang.RuntimeException

Class	Cause/Consequence
`ArithmeticException`	An attempt to perform an integer division by zero
`ArrayIndexOutOfBoundsException`	An attempt to access an array element using an index (subscript) less than zero or greater than or equal to the array's length
`NumberFormatException`	An attempt to convert a string that is not numeric to a number
`NullPointerException`	An attempt to use a `null` reference value to access an object
`NoSuchElementException`	An attempt to get a next element after all elements were accessed
`InputMismatchException`	The token returned by a `Scanner next...` method does not match the pattern for the expected data type

Summary of Exception Class Hierarchy



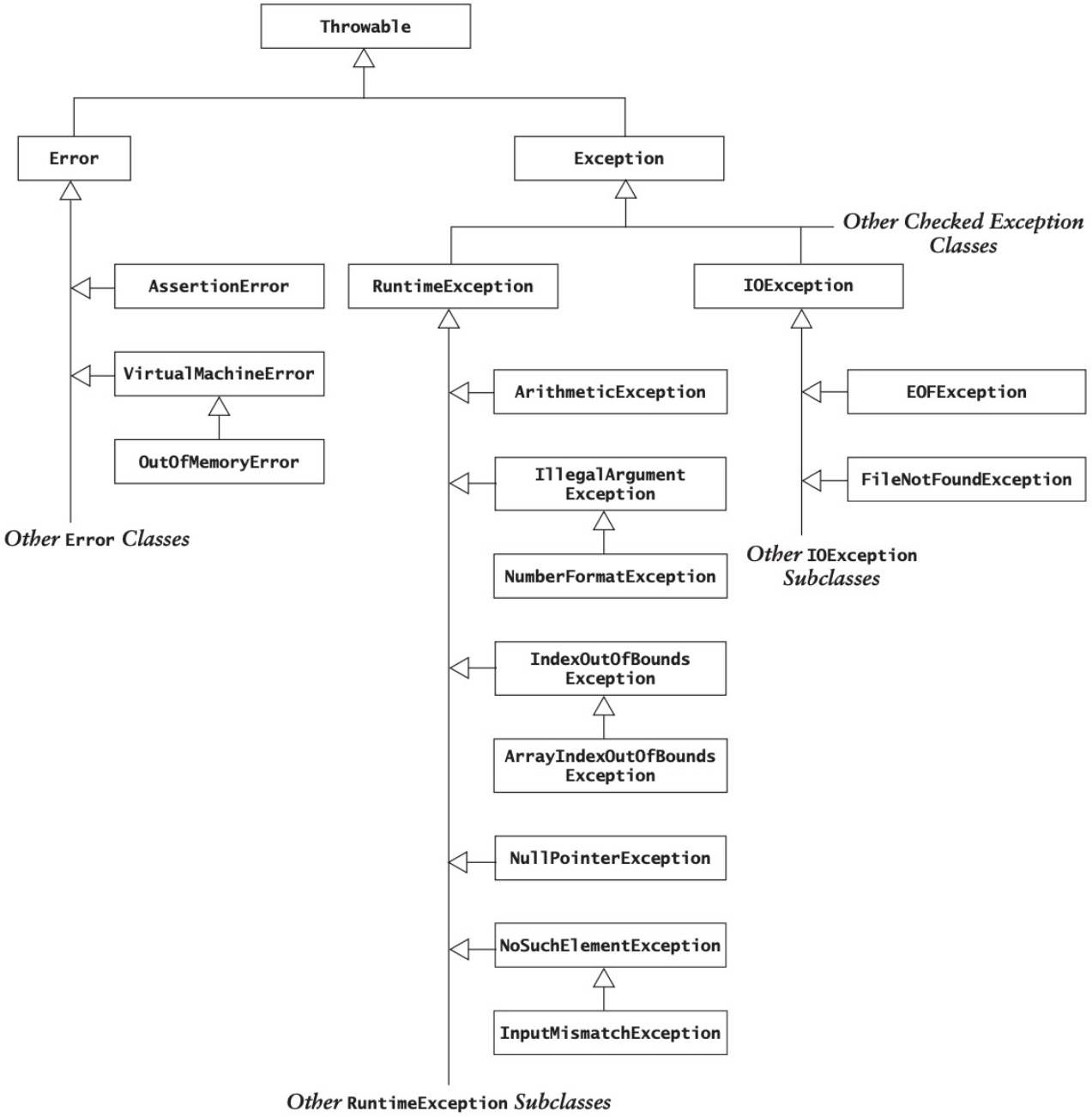
Summary of Commonly Used Methods from the `java.lang.Throwable` Class

Method	Behavior
`String getMessage()`	Returns the detail message
`void printStackTrace()`	Prints the stack trace to System.err
`String toString()`	Returns the name of the exception followed by the detail message

Class `java.io.IOException` and Some Subclasses

Exception Class	Cause
`IOException`	Some sort of input/output error
`EOFException`	Attempt to read beyond the end of data with a `DataInputStream`
`FileNotFoundException`	Inability to find a file

Exception Hierarchy Showing Selected Checked and Unchecked Exceptions



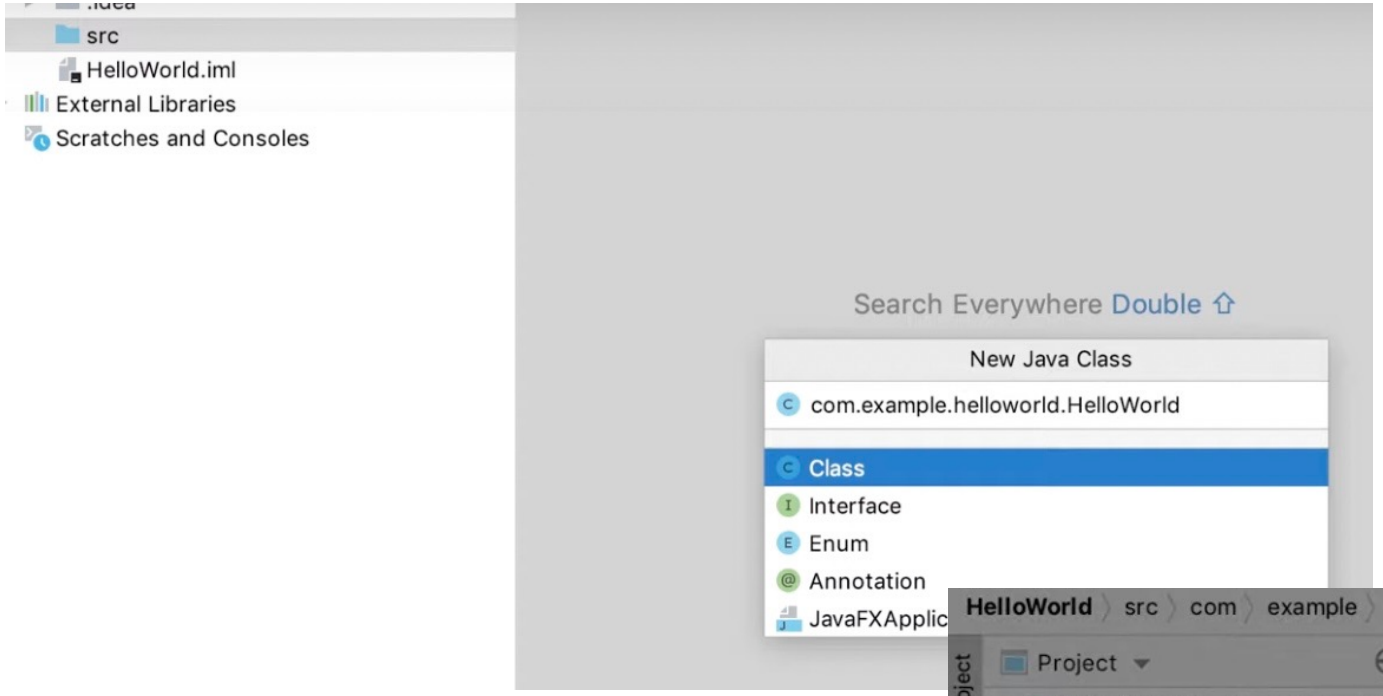
Packages and Visibility

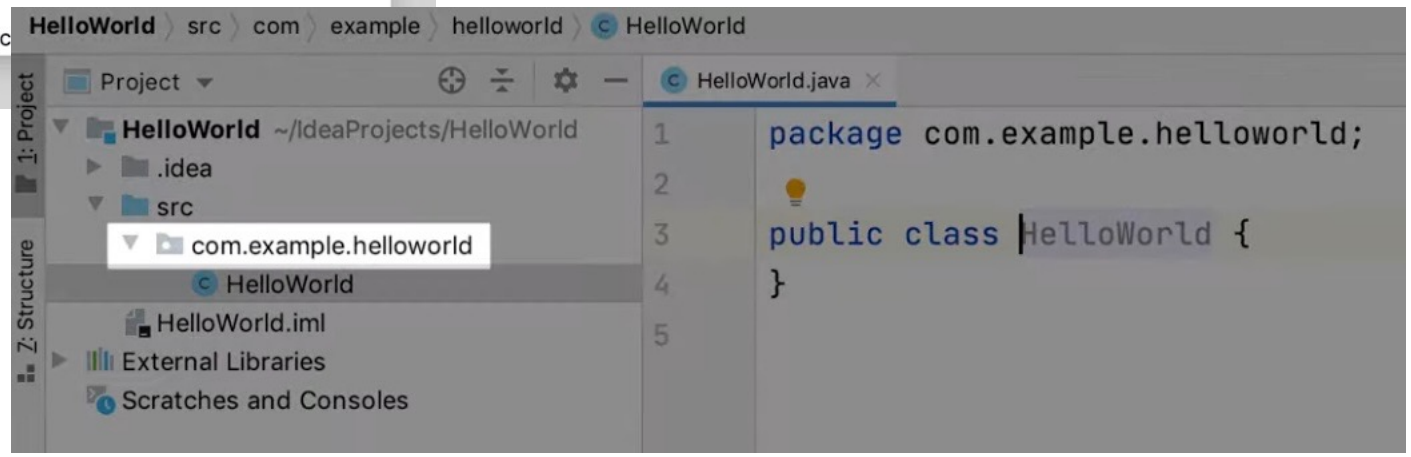
Packages

- The Java API is organized into packages such as java.lang, java.util, java.io, and javax.swing.
- The package to which a class belongs is declared by the first statement in the file in which the class is defined using the keyword **package**, followed by the package name.
- For example, we could begin each class in the computer hierarchy (class Notebook and class Computer) with the line:

```
package computers;
```

Create Package in IntelliJ IDEA





Packages and Visibility

- So far, we have discussed three layers of visibility for classes and class members (data fields and methods): private, protected, and public.
- There is a **fourth layer**, called package visibility, that sits between private and protected.
- Classes, data fields, and methods with package visibility are accessible to all other methods of the same package but are not accessible to methods outside of the package.
- By contrast, classes, data fields, and methods that are declared protected are visible within subclasses that are declared outside the package, in addition to being visible to all members of the package.
- Visibility Supports Encapsulation

Summary of Kinds of Visibility

Visibility	Applied to Classes	Applied to Class Members
`private`	Applicable to inner classes. Accessible only to members of the class in which it is declared	Visible only within this class
Default or package	Visible to classes in this package	Visible to classes in this package
`protected`	Applicable to inner classes. Visible to classes in this package and to classes outside the package that extend the class in which it is declared	Visible to classes in this package and to classes outside the package that extend this class
`public`	Visible to all classes	Visible to all classes. The class defining the member must also be public

A Shape Class Hierarchy

CASE STUDY Processing Geometric Figures

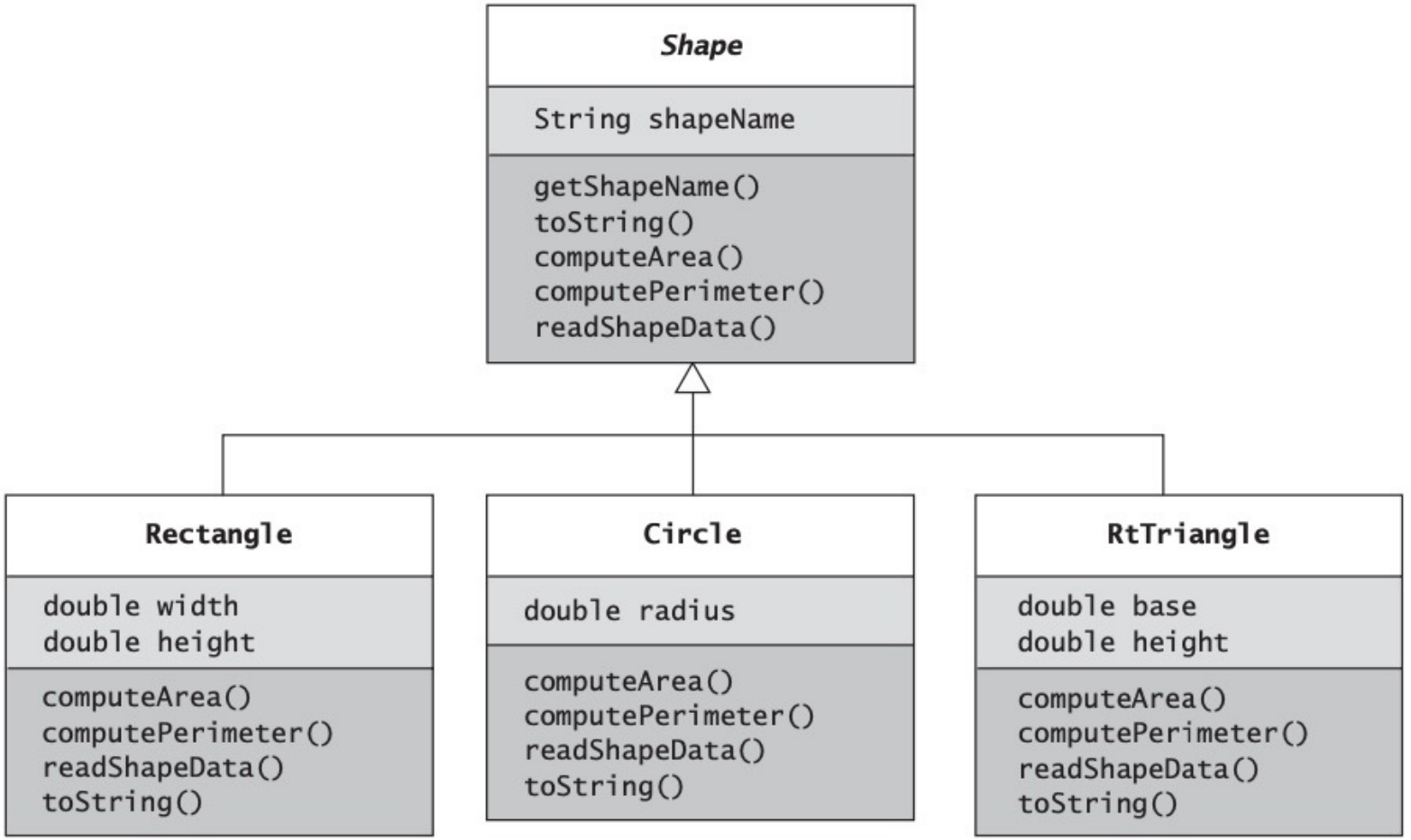
- Problem
  - We would like to process some standard geometric shapes.
  - Each figure object will be one of three standard shapes (rectangle, circle, and right triangle). We would like to be able to do standard computations, such as finding the area and perimeter, for any of these shapes
- Analysis
  - For each of the geometric shapes we can process, we need a class that represents the shape and knows how to perform the standard computations on it (i.e., find its area and perimeter).
  - These classes will be Rectangle, Circle, and RtTriangle.

A Shape Class Hierarchy

- Design
  - Class Rectangle has data fields width and height. It has methods to compute area and perimeter, a method to read in the attributes of a rectangular object (readShapeData), and a toString method.
  - The design of the other classes is similar

A Shape Class Hierarchy

Abstract Class Shape and Its Three Actual Subclasses



A Shape Class Hierarchy

Class `Rectangle`

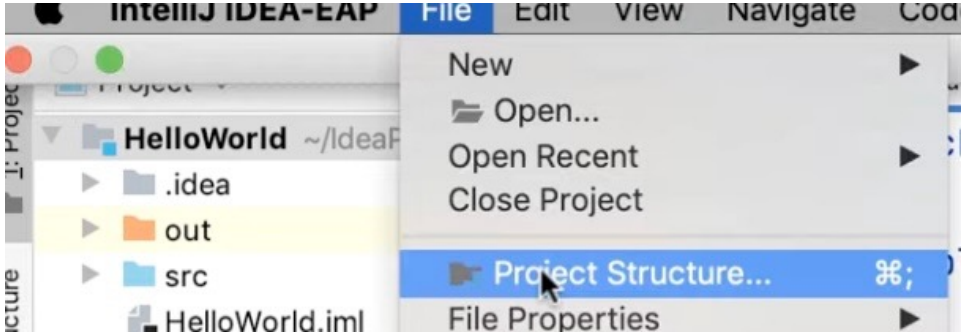
Data Field	Attribute
`double width`	Width of a rectangle
`double height`	Height of a rectangle
Method	**Behavior**
`double computeArea()`	Computes the rectangle area (width × height)
`double computePerimeter()`	Computes the rectangle perimeter (2 × width + 2 × height)
`void readShapeData()`	Reads the width and height
`String toString()`	Returns a string representing the state

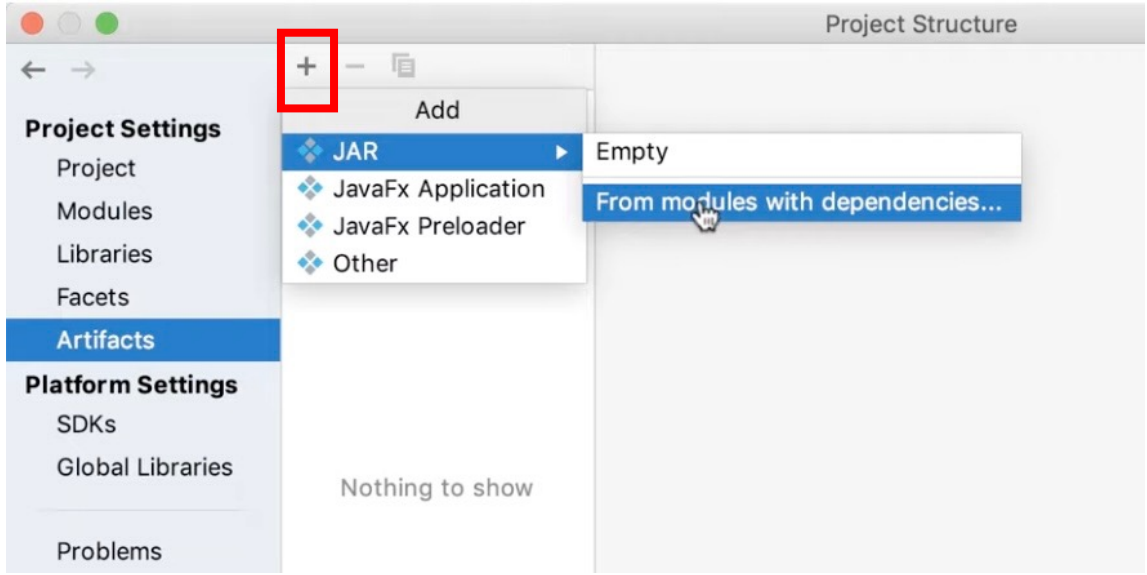
A Shape Class Hierarchy

- Next:
  - Implementation (code, abstract class Shape)
  - Testing

FYI Artifacts

- An artifact is an assembly of your project assets that you put together to test, deploy, or distribute your software solution or its part.
- In this example, we would like to create JAR (**Java ARchive**. It's a file format based on the popular ZIP file format and is used for aggregating many files into one)

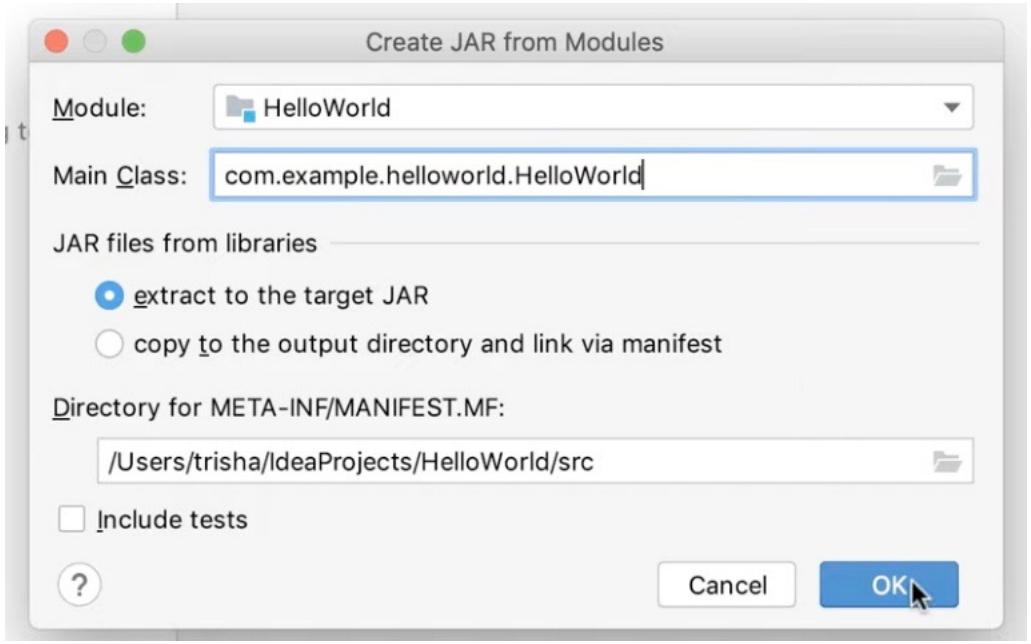


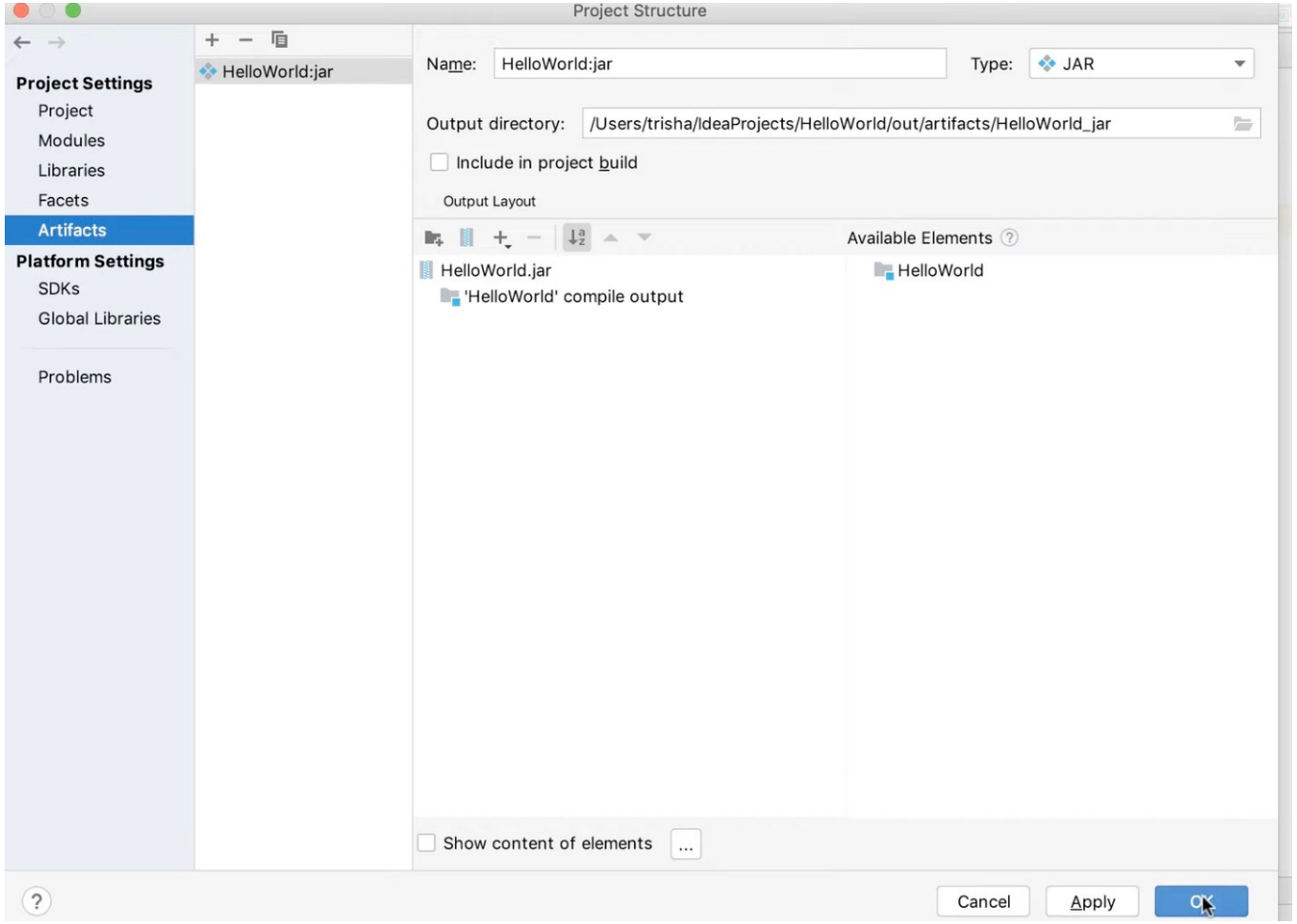


https://www.jetbrains.com/help/idea/working-with-artifacts.html

https://www.jetbrains.com/idea/guide/tutorials/hello-world/creating-a-package-and-class/

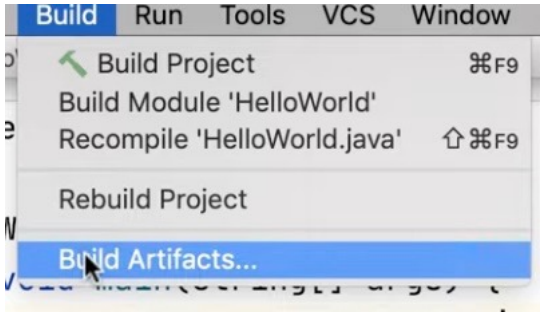
FYI Artifacts

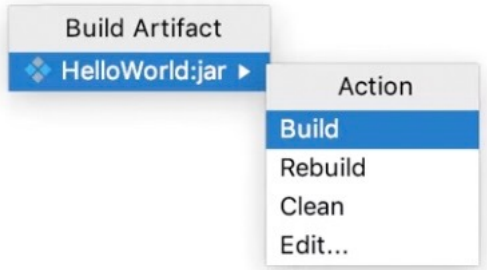


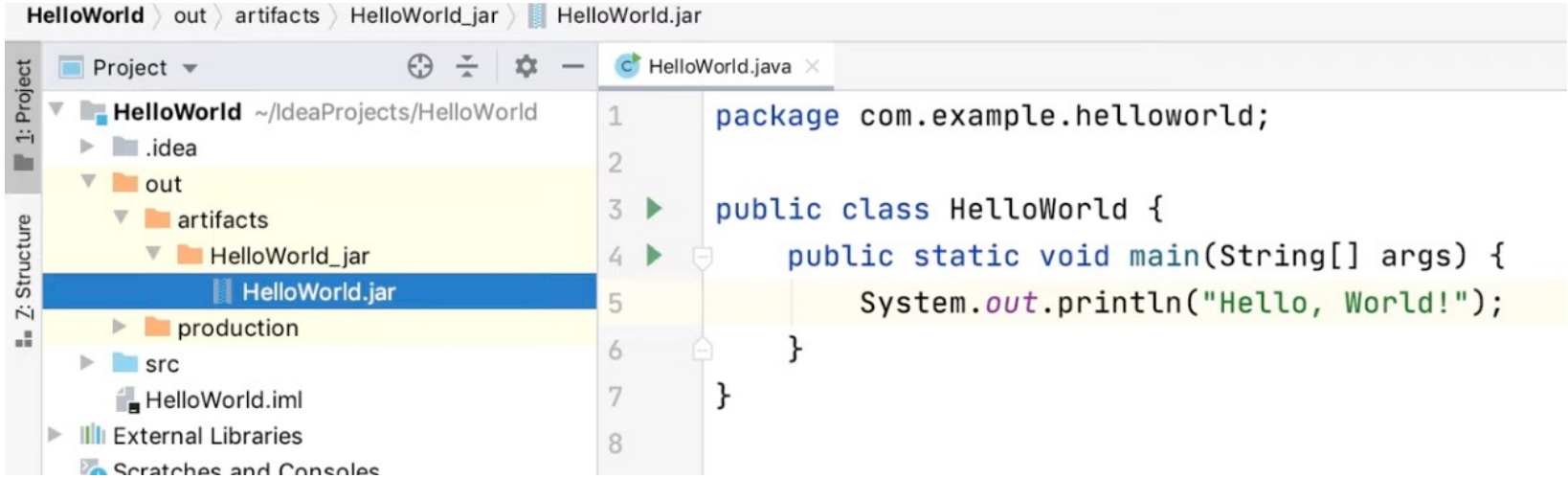


https://www.jetbrains.com/idea/guide/tutorials/hello-world/creating-a-package-and-class/

FYI Artifacts







Complete Tutorial Video: https://www.jetbrains.com/idea/guide/tutorials/hello-world/creating-a-package-and-class/

References

- Koffman, E. B., & Wolfgang, P. A. (2021). Data structures: abstraction and design using Java (4th Edition). John Wiley & Sons. [DSA]
- Weiss, M. A. (2010). Data Structures and Problem Solving Using Java (Fourth Edition). Addison-Wesley. [DSPS]
- Weiss, M. A. (2014). Data structures and algorithm analysis in Java (3rd Ed). Pearson.
- Karumanchi, N. (2017). Data Structures and Algorithms Made Easy In JAVA. CareerMonk.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data structures and algorithms in Java (6th Ed.). John Wiley & Sons.

Thank you