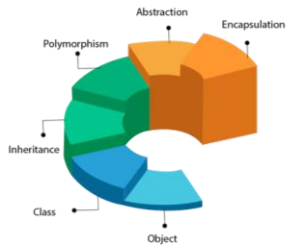
Name:		Date:	



TOPIC D Object Orientated Programming IB Computer Science (SL/HL)

Breakdown:

The section on Object Orientated Programming from the IB curriculum suggests you invest 30 hours of your time on this topic at SL and 45 hours at HL. This is broken down into these sections:

1) D.1 Objects as a programming concept (6 hours)
2) D.2 Features of OOP (4 hours)
3) D.3 Program development (20 hours)
4) HL Extension: D.4 Advanced program development (15 hours)

Syllabus Specifics:

To ensure you are fully prepared to take on the IB Computer Science external examinations (your exams!); be sure to read and understand the specific success criteria for this topic in each section.

Glossary:

You should remember to note down any new key terms you come across throughout the topics. Keep it updated to ensure you are understanding all the IB CS terminology required for the exams. You may add more as you feel fit.

Object	
Objects' actions	
Class	
Template	
Instantiation	
UML diagram	
Decomposition	
Inheritance	
Integer	
Real	
Boolean	
Parameter	
Encapsulation	

Polymorphism	
Encapsulatation	
Modularity	
Class	
Primitive	
Instance variable	
Paramter variable	
Local variable	
Method	
Accessor	
Mutator	

D.1.1 The general nature of an object

Exit skills. Students should be able to:[1]
Define the terms: object, objects' data and objects' actions. Describe the conceptual framework of objects in programming. Explain the use of objects as an abstract entity.

To start…

In the past programs were written procedurally the computer viewed and executed the code line by line, as programs became more complex this led to very long and complicated programs. The result is now we have object orientated programming; each object is a separate item which can be manipulated with its own behaviours. This allows us to make complex programs easier.

Learn it.

Person
DATA
Name
Height
Weight
Gender
Age
Eye Colour
Hair Colour
ACTIONS
Sleep
Wake up
Walk
Run
Climb stairs

Figure D.1: Person's Object Data and Actions

We begin this chapter by having a short conceptual discussion of the idea on an "object" before going into *object-oriented programming (OOP)*. To understand objects, we can bring an example from real life to help us along: people. Each person in the world might be different but they also have some traits which define them as humans. For example, a person looks a certain way; he/she has a height, weight, gender, age, eye/hair colour, etc. All these are general properties (data) that a person has. Furthermore, he/she also has the ability to perform certain actions, mainly, do things. For example, a person can sleep, wake up, walk, run, climb stairs, etc. Certainly, there might be some actions that some people can do that others cannot, like playing the piano, but there are some actions which common for all people. According to these data and actions, we construct Figure D.1 that depicts these components clearly.

An *object* is thus an *abstract entity* that describes the data that this entity has (a.k.a. properties or attributes) and the actions that this entity can perform (a.k.a. methods).

Complete the table of components for the abstract object "Vehicle"

Vehicle	
DATA	ACTIONS

D.1.2 Distinguishing between object and instantiation

Exit skills. Students should be able to:[1]
Define the terms: class, template and instantiation.
Distinguish between an object and instantiation.
Discuss memory use and code definitions that relate to object and instantiation.

We previously used the example of a Vehicle object. In OOP we have objects which can be manipulated; however, we need to first create a class for our objects to be created. One example of a class is a bicycle

class. Imagine we wanted to store details about all the bikes stored in a school. We would need a bike class that stores our knowledge of what a bike should have and do. We know that they speed up and slow down, this would be represented in Java using the following code:

```
public class bicycle{
      // the bicycle has two instance variables
      private int gear;
      private int speed;

      // there is a constructor class which allows us to make
      // a bicycle object we can manipulate

      public bicycle (int startGear, int startSpeed){
            gear = startGear;
            speed = startSpeed;
      }

      // the bicycle class has three methods
      public void setGear (int newGear){
            gear = newGear;
      }

      public void applyBreak (){
            speed = speed - 1;
      }
      public void speedUp(){
            speed = speed + 1;
      }
```

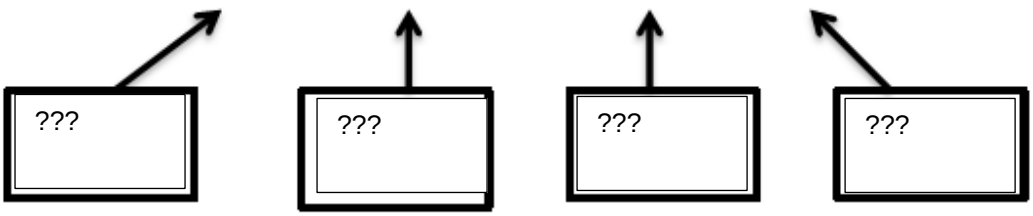
What is a class? Answer below with the help of these websites: https://www.educba.com/object-oriented-programming-in-java/ & https://www.w3schools.com/java/java_classes.asp

---

At the moment our bike will do nothing as it is only an idea. Its' only when we create an instance of our bike with a start speed and a starting gear do we have something tangible we can manipulate.

We must first "construct" the object using our class before we can begin to use it, see if you can complete the boxes below:

```
bicycle myBike = new bicycle (3, 10);
```



Activities

Check your understanding (theory):

1: What is the difference between a class and an object?

2: What is a constructor method?

3: Using the example above explain what the numbers between the ( ) are.

4: Think about a dog, what instance variables and methods would you need to create a dog?

Check your understanding (practical):

Critters are little creatures that can be adopted and grown into pets. They are brightly coloured creatures and each has a special power. The eat and drink like regular animals but they exercise by flying. An example critter is shown below:

Name: Katie
Colour: Blue
Special Power: Invisibility
Eats: popcorn
Drinks: Mountain Water

Implement the Critter class in java and use a main method test class to check it works.

Theory Support: https://bitly.im/5yhDJ Practical Support: Source Code

Answers to the theory questions (add more lines as necessary) →

Practical Java Code (screenshot):

Support: https://www.freecodecamp.org/news/java-object-oriented-programming-system-principles-oops-concepts-for-beginners/

D.1.3-4 UML Diagrams

Exit skills. Students should be able to:[1]
Define UML diagrams. Use UML diagrams to facilitate object design. Construct and interpret UML diagrams.

Activity: video

UML diagrams are very important in Java development. Watch the video https://www.youtube.com/watch?v=UI6lqHOVHic and make notes as you go so you can refer back to this later:

Define what a UML diagram is:

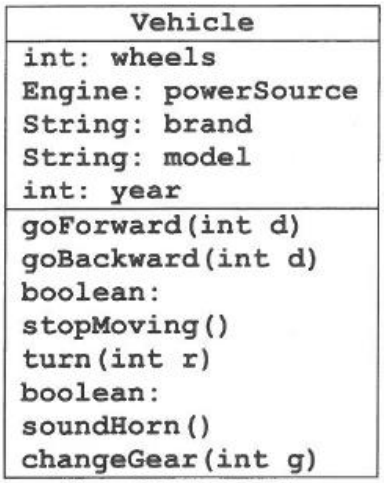


Figure D.5: Vehicle class

Taking the vehicle example from earlier, we have broken the class down into its 'data' and 'actions'.

Data are written like so: dataType: dataName

Actions are written like so: returnType: actionName (inputType). Input type is the data that is required for the action, whereas returnType is the data that will be returned by the method for example. When there is no returnType, this indicated a void return.

UML class diagrams show us what our classes contain, however, the can also show how the classes interact/collaborate with each other. Our programs are connected in various ways. We can use UML diagrams to illustrate this. See below the vehicle class and how this is connected to other classes.

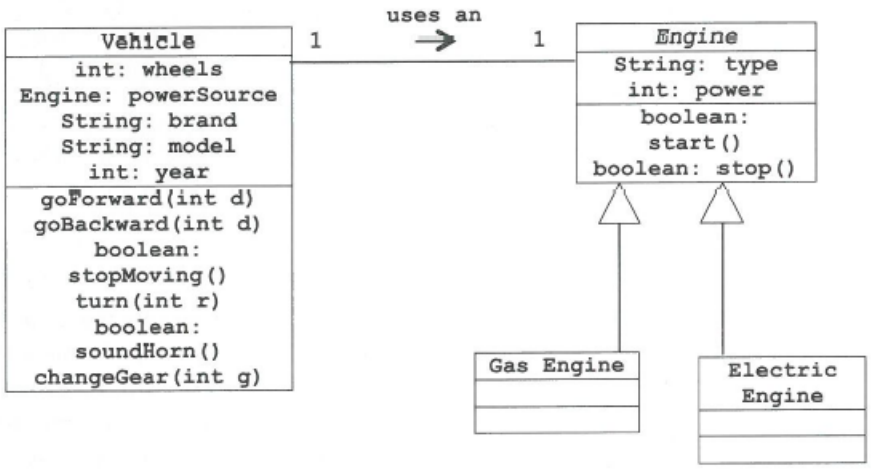


Figure D.6: UML class diagram

Here we see the Vehicle class using the Engine class (every Vehicle has an Engine). There are two types of engine. An arrow pointing towards the main Engine class suggests a relationship to the Engine class which means they both inherit the data and actions from the class. The arrows show an "Is a" relationship between the classes. There is no data within the Gas Engine or Electric Engine type, this is because of the inheritance (they both take the same data/actions).

Activity: Checking for understanding

What is inheritance? Complete the term in your glossary

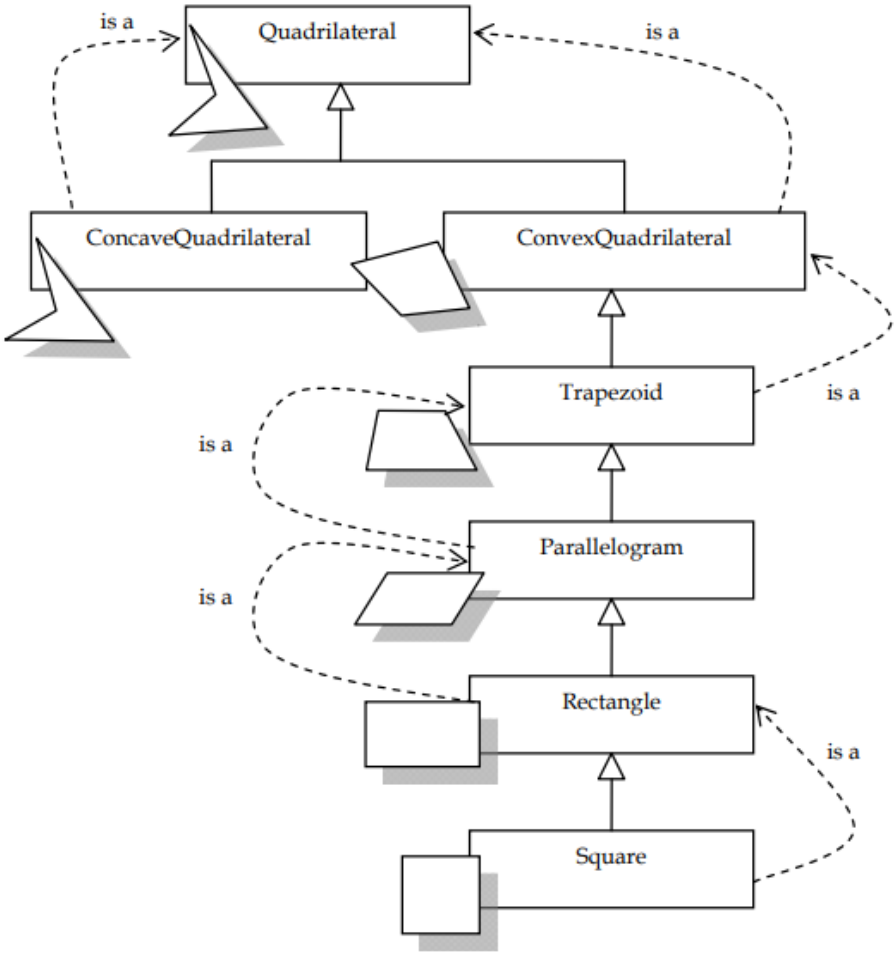
What are the advantages and disadvantages of this characteristic of OOP? Complete the table

Advantages	Disadvantages

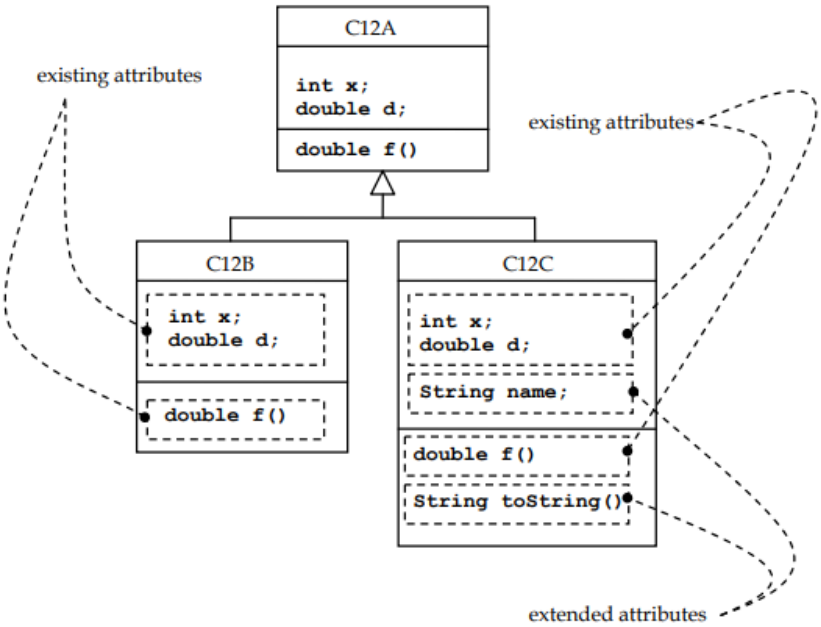
Learn it

An example of how inheritance might work:

A subclass can be inherited further by other classes. This makes a hierarchy of classes. The figure below shows an example of a hierarchy among some quadrilaterals.



In the style of a UML diagram



Let's take the previous "bicycle" example:

```
public class bicycle{
      // the bicycle has two instance variables
      private int gear;
      private int speed;

      // there is a constructor class which allows us to make
      // a bicycle object we can manipulate

      public bicycle (int startGear, int startSpeed){
            gear = startGear;
            speed = startSpeed;
      }

      // the bicycle class has three methods
      public void setGear (int newGear){
            gear = newGear;
      }

      public void applyBreak (){
            speed = speed - 1;
      }
      public void speedUp(){
            speed = speed + 1;
      }
```

So, we have our bicycle superclass. This contains all the common methods (features) of a typical bike. We can have more classes for each type of bicycle.

For example we have the mountain bike subclass

```
public class mountainBike extends bicycle {

      public int suspension;

  public mountainBike (int startSuspension, int
startGear, int startSpeed) {

      super (startGear, startSpeed);
      suspension = startSuspension;
      }

public int setSuspension (int susValue) {

      suspension = susValue;

      }
}
```

The variable that is unique to the mountain bike

The constructor method has the super and sub class start values

The variables for the superclass are sent to that constructor

This method is only for the sub class

For example we have the racing bike subclass

```
public class racingBike extends bicycle {

      public int tyreStyle;

  public racingBike (int startStyle, int startGear,
int startSpeed) {

      super (startGear, startSpeed);
      tyreStyle =startStyle;
      }

public int setTyre (int tyreValue) {

      tyreStyle = tyreValue;

      }
}
```

The variable that is unique to the racing bike

The constructor method has the super and sub class start values

The variables for the superclass are sent to that constructor

This method is only for the sub class

Activity: UML Class Diagrams

OK, using the classes above and the information below, draw a UML diagram to show how the classes are connected including their data and actions.

You should include the original superclass (Bicycle), and 3 subclasses (mountainBike, racingBike and stuntBike).

Data

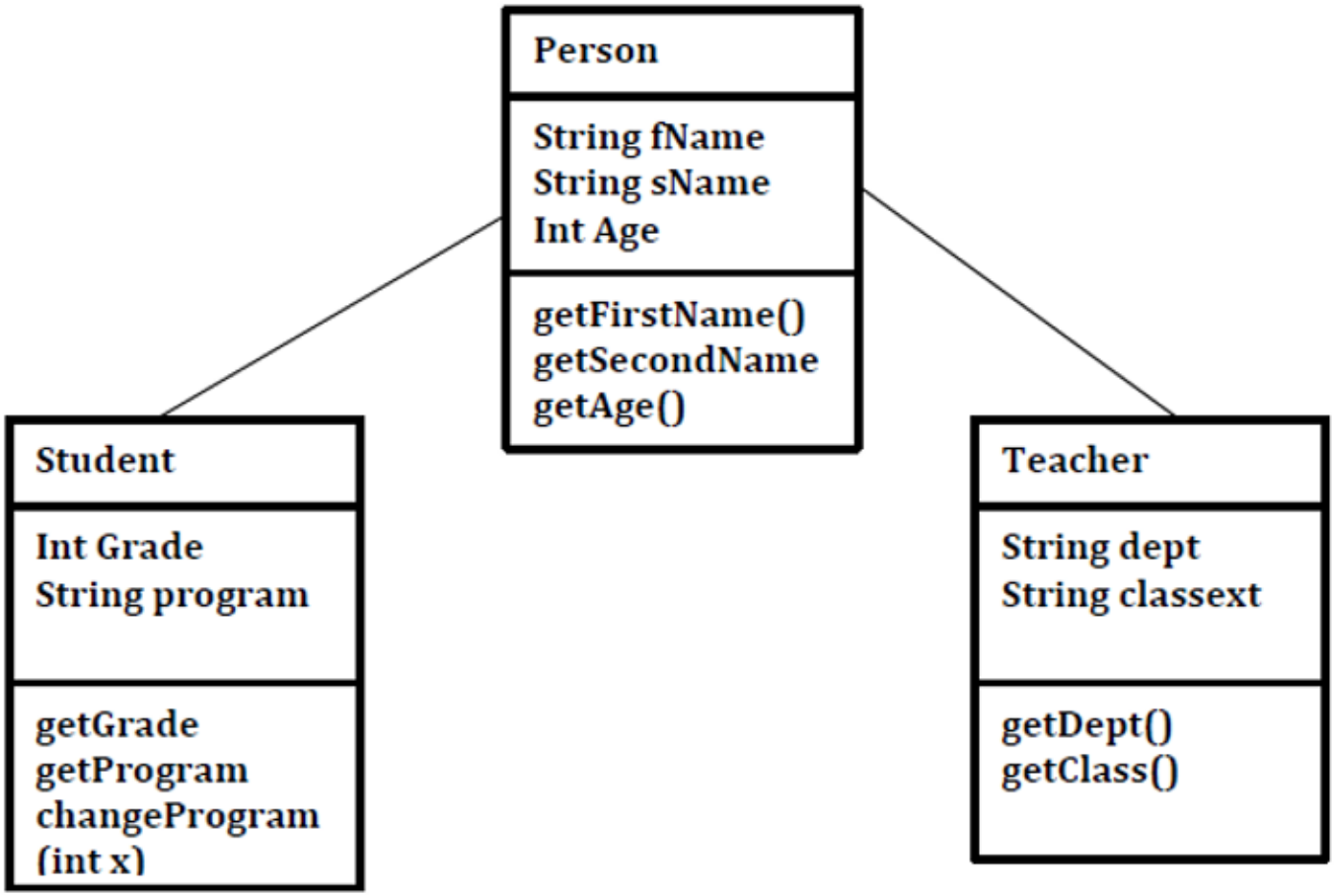
- ✓ The mountain bike has suspension
- ✓ The racing bike has wider tyres
- ✓ The stunt bike has adjustable seats

Actions (methods)

Using the above data, create actions for each of the data for the different bicycles.

Practical Activity

Using your amazing knowledge of inheritance (help: https://www.javatpoint.com/inheritance-in-java) create a new program in Java using the following UML diagram: (you should be using the main method only to check if it works)



You can see there is going to be 3 classes (Person, Student, Teacher). You may include all classes within the same Java file or you can choose to create 3 separate files (4 including the driver class). The latter is how you *should* be doing it ideally.

Side Note

Accessing class methods outside of the Java file you must use the keywords public and extends to show inheritance of the classes' objects as below:

Different Files

Bicycle.java

- public class bicycle

MountainBike.java

- public class mountainBike extends bicycle

Accessing classes from within the same java file is easier but not typically used once you develop your own skills. Notice the lack of public and extends keywords:

Same File

Bicycle.java

- public class bicycle
- class mountainBike

Paste your completed code here (screenshots or correctly formatted code) ask for password→

Try it yourself first. We will attempt it together afterwards.

Support? Checkout the source file: Source Code

Review – let's look at the source code

Learn it

UML Diagrams can illustrate how classes interact and what associations and inheritance they have. Here we see an association between Class A and Class B. These associations can also have multiplicities, which are placed at each end of the association line to signify the number of objects that one class has linked to another. Look at this generic UML diagram:

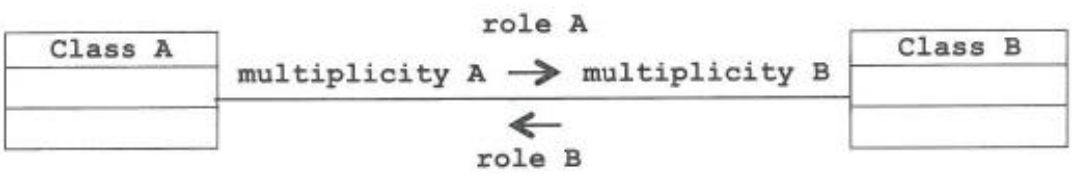


Figure D.7: Generic UML class diagram

Now, let's look at the previous Vehicle class:

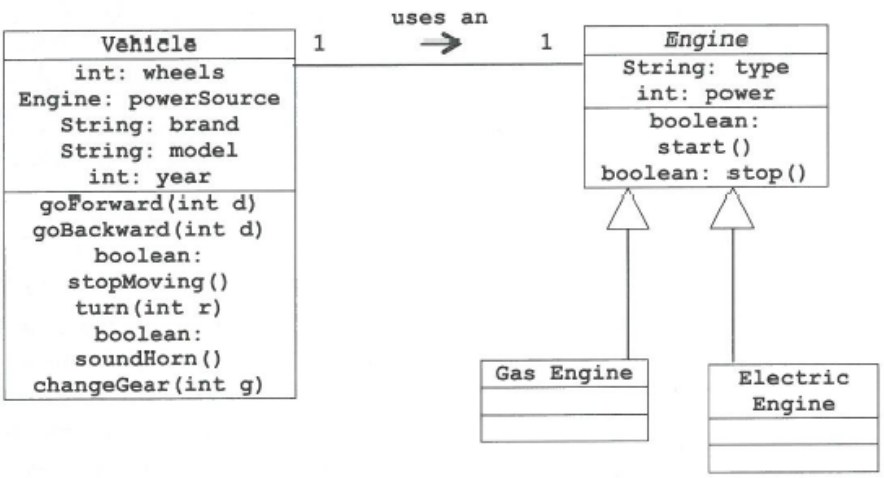


Figure D.6: UML class diagram

In this instance, the Vehicle class has an association with the Engine. It uses a multiplicity of 1 and 1 meaning that one Vehicle class can only be associated with one Engine class (because Vehicles only have one engine, right?). On the other hand, if we look at a company and draw the UML diagram per below:

Figure D.8: UML class diagram

We can see a multiplicity of 1 Company and 1..* Person, meaning for each Company class there can be many Person classes but a Person can only have one Company. Single numbers can also be used. For example, if you only wanted 11 people in a football team you could write 11.

Here is a quick reference list of multiplicities:

Common Multiplicities	
1	Exactly one
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Figure D.9: Common Multiplicities

Finally, you may be asked to draw a UML class diagram without the data or actions. This would typically be used to show an overview of the associations and multiplicities within a system. See below for the Vehicle Example again:

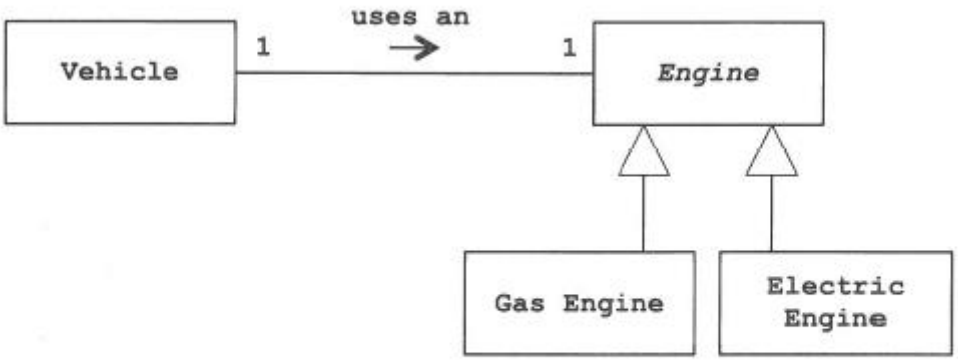


Figure D.10: UML class diagram

For further support on associations and multiplicities watch this
https://www.youtube.com/watch?v=BhEoV57nj0Q

Activities

Why might you use UML Class Diagrams?

1.

2.

3.

4.

Theory

Using the Java code below, draw a UML Class Diagram showing the different instance variables, actions(methods), associations, multiplicities of each class.
Support? https://www.javatpoint.com/uml-class-diagram

BuyLaptop

```
package ARP;

public class BuyLaptop {
        Customer customer;
        Laptop laptop;
        public String date ;
        public int qty;

        public BuyLaptop(Customer customer,Laptop laptop,String date,int qty)
        {
                this.customer=customer;
                this.laptop=laptop;
                this.date=date;
                this.qty=qty;
        }
        public void display()
        {
                customer.display();
                laptop.display();
                System.out.println("Date:"+date+"\nQuantity :"+qty);
                laptop.onService();
                laptop.offService();

        }

}
```

Customer

```
package ARP;

public class Customer extends User {
        private String email,cellNO;

        public Customer(String name,String loginID,String password,String email,String cellNO)
        {
                super(name,loginID,password);
                this.email=email;
                this.cellNO=cellNO;
        }
        public void display()
        {
                super.display();
                System.out.println("Email:"+email+"\nCell NO:"+cellNO);
        }

}
```

ElectronicGadget

```
package ARP;

public class ElectronicGadget {
        protected String menufacturer;

        public ElectronicGadget(String menufacturer)
        {
                this.menufacturer=menufacturer;
        }

        public void display()
        {
                System.out.println("Menufacturer :"+menufacturer);
        }
}
```

Laptop

```
package ARP;

public class Laptop extends ElectronicGadget {
        private String name,price,powerpack;

        public Laptop(String menufacturer,String name,String price,String powerpack)
        {
                super(menufacturer);
                this.name=name;
                this.price=price;
                this.powerpack=powerpack;
        }
        public void display()
        {
                super.display();
                System.out.println("Name:"+name+"\nPrice :"+price+"\nPowerpack :"+powerpack);
        }
        public void onService()
        {
                System.out.println("On Service :2 Years international Warranty, Battery and Charger 1
Year Warranty");

        }

        public void offService()
        {
                System.out.println("OFF Service :Not available");
        }

}
```

User

```
package ARP;

public class User {
        protected String name,loginID,password;
        public User(String name,String loginID,String password)
        {
                this.name=name;
                this.loginID=loginID;
                this.password=password;
        }

        public void display()
        {
                System.out.println("Name :"+name+"\nLogin ID:"+loginID+"\nPassword :"+password);
        }


}
```

D.1.5 Process of decomposition

Exit skills. Students should be able to:[1]
Describe the decomposition process of an object to several related objects. Explain how the decomposition process facilitates abstraction. Use the objects' decomposition process in real life situations.

What is decomposition? Watch the video and make notes: Decomposition in CS

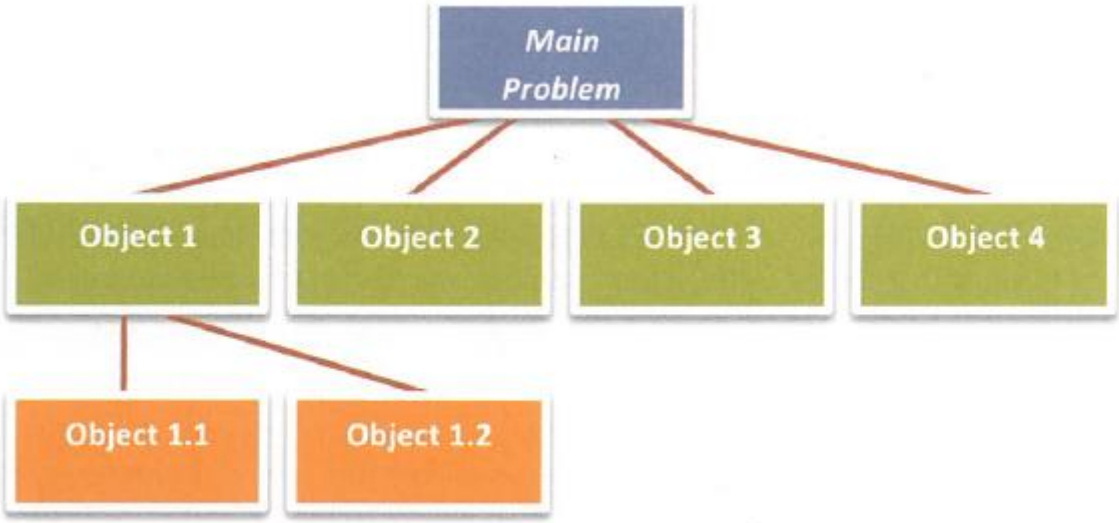


Figure D.11: The decomposition of a problem into related objects

Scenario:
An exam board uses a computer system to record examination entries, record marks for each exam, and reporting on results for each subject.

Details: Reporting on results involves collating results for each student in each centre (school or college) for which there are results, sending out certificates for each student showing exams passed and grades obtained, and preparing national statistics on how many students took each exam and the percentage of students gaining each grade in each subject.

Task: Draw a decomposed diagram like above to illustrate how this computer program may be broken down into sub-systems, which in turn may be made up of further subsystems.

here are some examples:

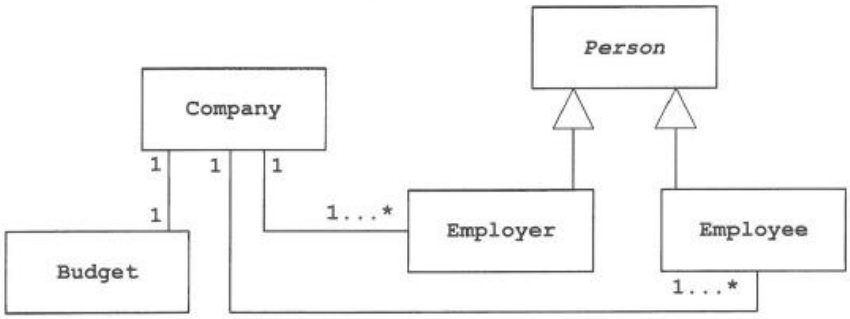


Figure D.12: The decomposition of a problem into related objects

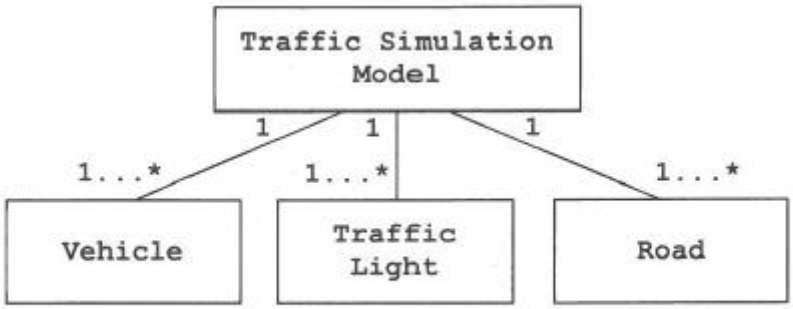


Figure D.13: The decomposition of a problem into related objects

use this to create your nice diagrams https://app.diagrams.net/ printscreen and paste here once finished

D.1.6 Relationships between objects

Exit skills. Students should be able to:[1]
Explain the dependency ("uses"), aggregation ("has a") and inheritance ("is a") relationship between objects in a given situation. Explain how the dependency ("uses"), aggregation ("has a") and inheritance ("is a") relationship facilitate abstraction.

Perform some independent research on the following terms and explain in as much detail as you can. You may use diagrams to aid your explanations.

Association

Dependency – "Uses" relationship

Aggregation – "has a" relationship

Inheritance – "is a" relationship

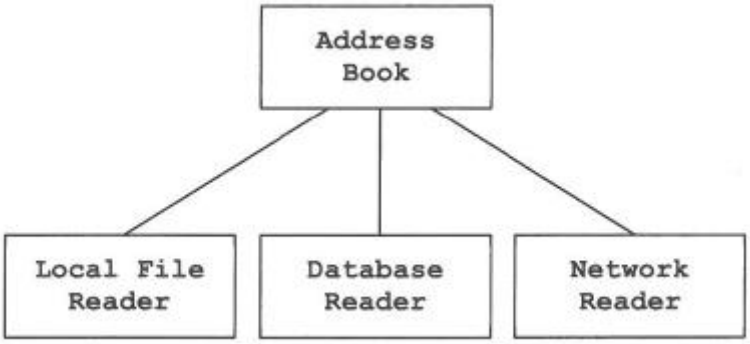
How can all the above facilitate abstraction?

Here is a very basic definition is you are not sure how to begin to answer this question:
https://www.bbc.co.uk/bitesize/guides/zttrcdm/revision/1

D.1.7 Need to reduce dependencies between objects

Exit skills. Students should be able to: [1]
Explain the negative effects that unnecessary dependencies between objects cause. Discuss the increase of maintenance overheads because of increased dependencies

Dependencies happen when objects rely on one another. When one object is used, the other must also be used. These dependencies are directional, so one object may depend on another but not the other way around.



A simple line shows the dependencies between and Address Book and the other objects that are required to make the program run. What issues could arise if the Network Reader object changed in the way it interacted with the Address Book?

How could the developer solve the issue of changed to a single object that depends on the rest of the program/code? Explain below:

D.1.8 Constructing Related Objects

Exit skills. Students should be able to:[1]
Develop objects for a given scenario. Develop various object definitions. Explain the relationships of objects to each other and to any additional classes defined by a given scenario.

Scenario (study carefully)

A University wishes to have an online library management system developed to keep track of the different books it has, the users (students + staff) of the system, their account details, the librarians that work there as well as the library database for adding, deleting, searching, and updating the book list.

Activity

Part 1: Develop a top down design breakdown of objects and dependencies between the objects like we saw in fig D.11 on page 17. If you need support, see the coursebook (p.307) for an idea of how to organise it.

Part 2: Present your breakdown to the scenario above as a unified modelling language (UML) diagram. If you need support, see the coursebook (p.308) for an idea of how to create UML diagrams that depict dependencies between objects.

Finished? State the relationships between the objects in your UML diagram

D.1.9 Data Types

Exit skills. Students should be able to:[1]
Explain the need of integer, real, string and Boolean data types. Explain how real world items are represented, store and manipulated by different data types.

Activity

A very simple section… Complete the table of data types and state why they are used along with some examples of actual data using the specific type:

Type	Why it is used	Example

Finished: What are primitive data types? Explain then give the 4 different primitive data types we find in the Java language

Primitive Data Types in Java:

1.
2.
3.
4.

D.1.10 Data items passed as parameters

Exit skills. Students should be able to:[1]
Define the term parameter. Explain the use of parameters. Explain the pass-by-value process. Explain how data items are passed to and from actions (methods in Java) as parameters.

Using the book (p.310-12)

Answer the questions:

What is a parameter?

Why do we use parameters?

Explain the pass-by-value process

How are data items passed to and from methods as parameters?

Practical Activity

Using the below pseudocode, create a simple UML diagram highlighting the classes, attributes and methods needed to create the program in Java

```
NUMBER1 = 3
NUMBER2 = 4
CAL = new Calculator()
NUMBER3 = CAL.increment(NUMBER2)
output "NUM1: ", NUMBER1, "NUM2: ", NUMBER2, "NUM3: ", NUMBER3
RESULT1 = CAL.add(NUMBER1, NUMBER2)
RESULT2 = CAL.add(NUMBER1, NUMBER3)
output "RESULT1: ", RESULT1, "RESULT2: ", RESULT2
```

Output:

```
NUM1: 3 NUM2: 4 NUM3: 5
RESULT1: 7 RESULT2: 8
```

Insert your UML diagram here:

Screenshot/paste your working code here once you have finished in the IDE:

Projects (SL)

Project One – The SuperHero Decider
IF Statements

Instructions:

It is a well known fact that all super heroes have unique characteristics. The following flow chart shows the unique character traits of each super hero. What you have to do is create a program that will ask the user questions and depending on their answers show them which super hero they should be and their characteristics.

Project Two – Password Protector
Loops
Instructions:

While loop (conditional loop)
The while loop allows us to repeat until a condition is met. Here is the pseudocode for a password protection program. Implement this program in Java.

```
Create password variable
Create userInput variable
Create found variable and set to false
Create tries variable

Start loop
        Get user input
                If userInput = password
                        Display enter message
                        Found = true
                End if
End loop when tries > 3 or found = true
                        Display failed message
```

Extension Task (higher level): create a password list in a text file. Compare the user input to the text file. If the password appears in the field then allow access.

For Loop (unconditional loop)
The for loop allows us to repeat a section of code as many times as we tell it too. Here is the pseudocode for entering data into an array.

```
Create integer [10] testMarks
for each item in array
        create temp integer
        temp = user input
        current item in array = temp
end loop
```

Extension Task (HL and SL): Find the highest test result and display it on screen, find the lowest test result and display it on screen. Calculate the average test result and display it on screen.

Project Three – The Alias Generator
Arrays

Instructions:

Sometimes people like to remain anonymous and its harder than you think to come up with an alias. Therefore you should create an alias generator for creating anonymous names.

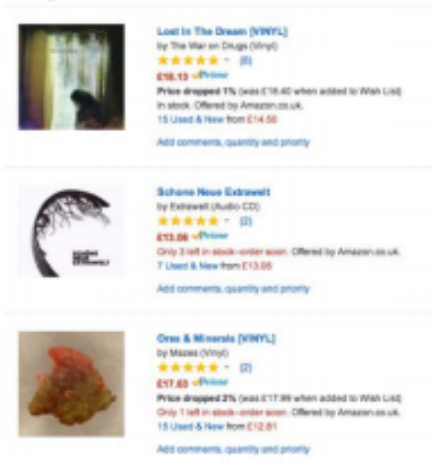
The outline pseudocode for this task would be:

```
Declare firstname array [10]
Declare surname array [10]

Add 10 names to firstname array
Add 10 names to surname array

Loop
        Generate random firstname
        Generate random surname
        Display name
Until
        User presses x
```

* note this program does make use of the random number generator which is part of the maths class *



Project Four – Amazon WishList

Arrays of classes

Instructions

Amazon allows you to make a wish list of items that you would like to buy. This acts as a reminder for when you feel like buying something or you can send it to people for Christmas / Birthdays etc.

You should create a similar idea. Create an "item" class, see the object diagram below, then allow people to enter the items they would like to purchase at a later date.

Once they have entered their items they should be printed out in a list on screen so people can view them easily.

Item
Int id String desc String dept Double cost String message
Int getID() String getDesc() String getDept() Double getCost() String getMessage()

Main Method:

Create array

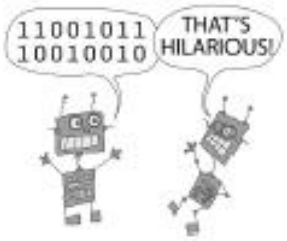
For each item in the array
    Enter data
    Create a new instance of the item class
End loop

For each item in the array
    Print the name and description of each item

Extension task – if you are HL you could use an array list and let the user store as many items as they like. This could also be saved to file for use at a later date.

HL

Project Two – Binary Converter



Binary to Decimal and Back Converter – Create a convertor that changes binary numbers into decimal numbers. To make the program even more useful the program can change decimal numbers into binary. If you are unsure how these numbers are connected – ask!
Tools recommended: An algorithm and lots of IF statements.

Project Three – Distance Calculator

Distance Between Two Cities – Calculates the distance between two cities and allows the user to specify a unit of distance. I would limit this to major cities in one country to start.
Tools recommended: An 2D array would probably be a good idea.

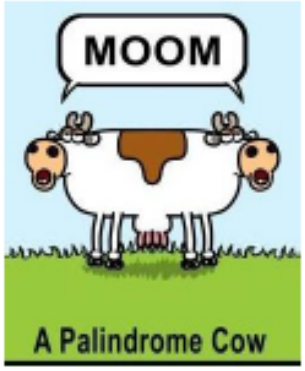
Traveltrade.visitscotland.org

Project Four – Pig Latin

Pig Latin – Pig Latin is a game of alterations played on the English language game. To form the Pig Latin form of an English word the initial consonant sound is transposed to the end of the word and an ay is affixed (Ex.: "banana" would yield anana-bay). Read Wikipedia for more information on rules.
Tools recommended: the string class

Project Five – Vowel Counter

Count Vowels – Enter a string and the program counts the number of vowels in the text. For added complexity have it report a sum of each vowel found.
Tools recommended: the string class.



Project Six – Palindrome Checker

Check if Palindrome – Checks if the string entered by the user is a palindrome. That is that it reads the same forwards as backwards like "racecar"
Tools recommended: the string class

Project Seven – Fortune Teller

Fortune Teller (Horoscope) – A program that uses objects to randomly generate a horoscope. Remember that horoscopes should be nice and generic so everyone thinks it relates to them.
Tools recommended: a linked list or similar.

Project Eight – Random Gift Suggestor

Random Gift Suggestions – Enter various gifts for certain people when you think of them and when its time to give them a gift (xmas, birthday, anniversary) it will randomly pick one and perhaps places you can get it.
Tools recommended: A linked list or similar.

Project Nine – Recipe Manager

Recipe Creator and Manager – Create a recipe class with ingredients and a put them in a recipe manager program that organizes them into categories like deserts, main courses or by ingredients like chicken, beef, soups, pies etc.
Tools recommended: A linked list or similar.

Project Ten – Vending Machine

Vending Machine – Create an application which takes money and dispenses various types of candy or other item. The user enters a number and letter sequence, like D9, and have it return an instance of "Item" which of the proper type. Example when they press D9 it will return a type of candy bar which is an instance of dairy milk.
Tools recommended: A linked list or similar and 2D arrays.

Project Twelve – Family Tree Creator

Family Tree Creator – Create a class called "Person" which will have a name, when they were born and when (and if) they died. Allow the user to create these Person classes and put them into a family tree structure. Print out the tree to the screen.
Tools recommended: A linked list or similar and files.

Project Thirteen – Doctor Scheduler

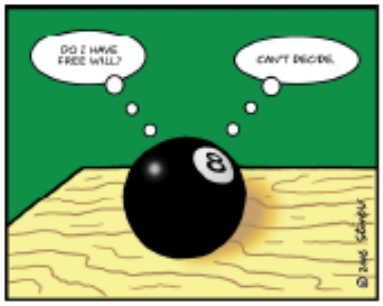
Patient / Doctor Scheduler – Create a patient class and a doctor class. Have a doctor that can handle multiple patients and setup a scheduling program where a doctor can only handle 16 patients during an 8 hr work day.
Tools recommended: A linked list or similar and files.

Project Fourteen – The Password Safe

Password Safe – A program which keeps track of passwords for sites or applications and encrypts them with a key so that no one can read them.
Tools recommended: A linked list or similar.

Project Nineteen – Magic Eight Ball

Magic Eight Ball – Create a file full of random sayings and answers. Then have the magic eight ball pick one at random in response to a user's question. Allow the user to enter the question and then show them the randomly picked answer.
Tools recommended: A linked list or similar and files.



Abstract Data Types – Projects

Check Your Understanding

1. Implement the program above paying particular attention to how the iterator works.
2. The linked List Quiz – how pirate are you?

On Facebook you constantly see updates such as "if I was a character from sesame street I would be the Count" or "the city I should definitely live in is Paris". In this task you are going to create a program that tests how pirate someone is. In an advanced data structure you will store questions. Each will have 4 possible choices and each choice has a points value. You should keep a running total of the points and at the end depending on how many points the person has display how pirate they are. (the question are on the next page)

Class Diagram:

Question
String Question String answerA String answerB String answerC String answerD Int valueA Int valueB Int valueC Int valued
getQuestion() getAnswerA() getAnswerB() getAnswerC() getAnswerD() getValueA() getValueB() getValueC() getValueD()

Program Outline:

```
1:  Create Linked List
2:  Populate linked list
3: for each question
        display question
        get answer
        get value of answer
        add value to current total
4:  display pirate worthiness
        if value is > 30                    if value <25 and > 18
                display very message                display average message
        if <30 and > 25                     if value < 18
                display high message                display not at all message
Questions value of answers in brackets:
```

1: what is your preferred pet?
- Parrot (5)
- Cat (4)
- Dog (3)
- Unicorn(1)

2: What is your favourite color?
- Black (3)
- Silver (4)
- White (1)
- Gold (5)

3: What is your preferred mode of transport?
- Car (1)
- Bike (3)
- Horse (4)
- Boat (5)

4: What is your preferred hair style?
- Long (5)
- Short and neat (1)
- Dyed (3)
- Curly (2)

5: What would be your preferred fortune?
- Clothes (1)
- Geeky Stuff (2)
- Gold (5)
- Car (3)

6: How do you say "hello"?
- Ahoy (5)
- Hola (3)
- Good day (1)
- Hi (2)

> 30
You are indeed a pirate, head out on your boat, raise your skull and crossbones and sail off into the sunset.
<30 and >25
you are well on your way to becoming a pirate, with a bit more training you could be spending your life at sea.
<25 and > 18
You are average, you dream of the sea but you are far too nice to live it. Maybe consider the wanderlust lifestyle
<18
You couldn't be less of a pirate if you tried and that is ok the world needs good people too!

Extra - Binary Trees (using OOP)

Tree implementation is quite complex however here is code that will allow someone to add, remove and search a tree

```
public class Node {

    public Comparable data;
    public Node left;
    public Node right;

    public void addNode(Node newNode){
        int comp = newNode.data.compareTo(data);
        if (comp <0){
            if (left == null) left = newNode;
            else left.addNode(newNode);
        }
        if (comp > 0){
            if (right == null) right = newNode;
            else right.addNode(newNode);
        }
    }
    public void printNodes(){
        if(left != null)
            left.printNodes();
            System.out.print(data + " ");
        if (right != null)
             right.printNodes();
    }
}
```

```
public class BinarySearchTree {

    private Node root;

    public BinarySearchTree()
    {
        root = null;
    }

    public void add (Comparable obj){
        Node newNode = new Node();
        newNode.data = obj;
        newNode.left = null;
        newNode.right = null;
        if(root == null) root = newNode;
        else root.addNode(newNode);
    }

    public boolean find (Comparable obj){
        Node current = root;
        while (current != null){
            int d = current.data.compareTo(obj);
            if (d ==0 ) return true;
            else if (d>0) current = current.left;
            else current = current.right;
        }
        return false;
    }
```

```
public void remove(Comparable obj){
    Node toBeRemoved = root;
    Node parent = null;
    boolean found = false;
    while (!found && toBeRemoved != null){
        int d = toBeRemoved.data.compareTo(obj);
        if (d==0) found = true;
        else{
            parent = toBeRemoved;
            if(d >0) toBeRemoved = toBeRemoved.left;
            else toBeRemoved = toBeRemoved.right;
        }
    }
    if (!found) return;

    if (toBeRemoved.left == null || toBeRemoved.right == null){
        Node newChild;
        if (toBeRemoved.left == null)
            newChild = toBeRemoved.right;
        else
            newChild = toBeRemoved.left;

        if (parent == null)
            root = newChild;
        else if (parent.left == toBeRemoved)
            parent.left = newChild;
        else
            parent.right = newChild;
        return;

    }


        }
        Node smallestParent = toBeRemoved;
        Node smallest = toBeRemoved.right;
        while (smallest.left != null){
            smallestParent = smallest;
            smallest = smallest.left;
        }
        toBeRemoved.data = smallest.data;
        if (smallestParent == toBeRemoved)
            smallestParent.right = smallest.right;
        else
            smallestParent.left = smallest.right;
    }
    public void print(){
        if(root!=null)
            root.printNodes();
        System.out.println();
    }

}
```

Driver method

```
    public static void main(String[] args) {

        BinarySearchTree mytree = new BinarySearchTree();
        Scanner myScanner = new Scanner(System.in);

        // adds some things to the tree and then prints out the tree
        mytree.add("linzie");
        mytree.add("hates");
        mytree.add("pineapples");


        // do a comparison to see if something exists in the tree and print out an appropriate message
        System.out.println("what would you like to exist?");
        String userInput = myScanner.next();

        if (mytree.find(userInput)){
            System.out.println("it does indeed exist the world is a better place");
        }
        else {
            System.out.println("no i am sorry but here is the list");
            mytree.print();
        }

        mytree.remove("hates");
        mytree.print();
    }

}
```