COMP122 Week 6

Inheritance

UNIVERSITY OF LIVERPOOL

Dr. Patrick Totzke
totzke@liverpool.ac.uk

https://liverpool.instructure.com/courses/59716

Wikipedia on Inheritance

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item

Inheritance (object-oriented programming)

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)* [show]

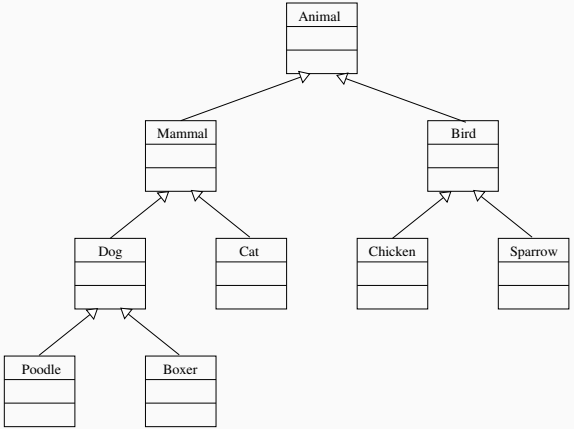
In object-oriented programming, **inheritance** is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation. Also defined as deriving new classes (sub classes) from existing ones (super class or base class) and forming them into a hierarchy of classes. In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object (except: constructors, destructor, overloaded operators and friend functions of the base class). Inheritance allows programmers to create classes that are built upon existing classes,[1] to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a directed graph. Inheritance was invented in 1969 for Simula.[2]

An inherited class is called a **subclass** of its parent class or super class. The term "inheritance" is loosely used for both class-based and prototype-based programming, but in narrow use the term is reserved for class-based programming (one class *inherits from* another), with the corresponding technique in prototype-based programming being instead called *delegation* (one object *delegates to* another).

Inheritance should not be confused with subtyping.[3][4] In some languages inheritance and subtyping agree,[a] whereas in others they differ; in general, subtyping establishes an is-a relationship, whereas inheritance only reuses implementation and establishes a syntactic relationship, not necessarily a semantic relationship (inheritance does not ensure behavioral subtyping). To distinguish these concepts, subtyping is also known as *interface inheritance*, whereas inheritance as defined here is known as *implementation inheritance* or *code inheritance*.[5] Still, inheritance is a commonly used mechanism for establishing subtype relationships.[6]

Inheritance

. . . describes the “is a” relationship between one class (called *superclass*) and *derived* classes (called *subclasses*) which *inherit* the characteristics from the superclass. Subclasses can add new attributes/methods and also *override* existing methods.



Person/Student/Lecturer example

A `Student` is a `Person` but with additional information such as a student number, program of study, and year they are in.

All methods of `Person` are still available.

Often, you would first first define, implement and test a `Person` class and later define the `Student` class that extends it. When testing the subclass one only needs to deal with the new attributes/methods.

Advantages of Inheritance

- Inheritance can introduce more abstraction in the code.
- It enhances code re-use.
- It improves the code readability.
- Properly applied, inheritance can reduce software maintenance costs.

Example: Club Membership

Design and implement a Java program that stores details about club members, e.g. their name and club ID number.

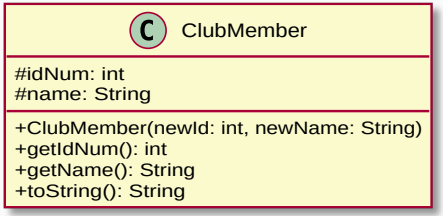
Additionally, some of the club members are committee members.

Committee members have all the attributes and behaviour of a club member, but additionally need their position on the committee (e.g. treasurer, chair, secretary, etc) to be stored.

Club Example – The ClubMember Class

First, we give a class that describes the (generic) members of the club, each with attributes `idNum` and `name`.

This looks like classes we have given elsewhere, i.e. this class has some attributes, a constructor, and some methods.



Note we have declared the attributes as `protected` (using "#"). This means that the attributes will be visible in any class that extends this one. (More on this later...)

Implementation of ClubMember

```
1  public class ClubMember {
2
3      /* Attributes */
4     protected int idNum;
5     protected String name;
6
7     /* Constructor */
8     public ClubMember(int newID, String newName) {
9        idNum = newID;
10       name = newName;
11    }
12
13    /*  Other methods */
14    public int getIdNum() { return idNum; }
15    public String getName() { return name; }
16
17    public String toString() {
18       return("Member Name: " + getName() + " ID number: " + getIdNum());
19    }
20 }
```

Club Example – Comments on ClubMember

- The class `ClubMember` is a very simple class with two attributes, one constructor, two get methods, and the `toString()` method.
- A `protected` identifier can be accessed from code in subclasses whereas a `private` identifier is only accessible in the class where it is defined.

We could add additional methods, such as a mutator `setName(...)` in case someone changes their name, but for now we will leave this class definition as it is here.

Club Example – CommitteeMember

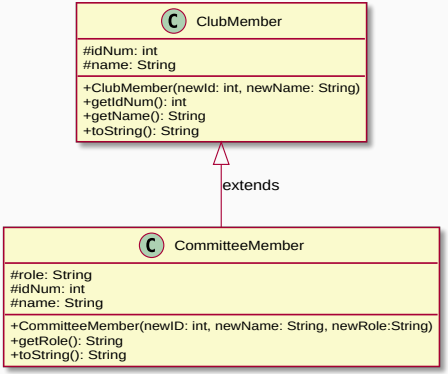
We now declare a class `CommitteeMember` that *inherits* the attributes and methods of `ClubMember` but *also* allows extra attributes and methods.

This adds a new attribute `role` which describes the role the member plays on the committee, and a corresponding getter.

We also redefine (override) the `toString()` method.

In the class diagram, we denote inheritance using an open arrow. (I am also writing the word "extends" here to emphasize this.)

Club Example – in UML



Syntax for Subclass Definitions

The Java keyword `extends` is used to specify the subclass/superclass relationship in Java class file definitions.

```
1  modifiers class SubClass extends SuperClass {
2  ...
3  }
```

where modifiers are optional (this could be `public` or `private`) and `SubClass` and `SuperClass` are valid identifiers for the (sub)class and superclass.

Club Example – The Subclass CommitteeMember

```
1  public class CommitteeMember extends ClubMember {
2
3      /* additional attribute */
4    protected String role;
5
6    /*  Constructor */
7    public CommitteeMember(int newID, String newName, String newRole) {
8       super(newID, newName);
9       role = newRole;
10   }
11
12   /*  getter for new atribute */
13   public String getRole() { return role; }
14
15   /* override toString method*/
16   public String toString() {
17      return("Committee Member Name: " + getName() +
18              "  ID number: " + getIdNum() + " Role: " + getRole());
19   }
20 }
```

Club Example – Comments on CommitteeMember

- The keyword `extends` specifies that the class is derived from the class `ClubMember`.
- The keyword `super` calls the superclass constructor. If used, it must be the first statement in a constructor, otherwise you get a syntax error.
  `super` can also be used to access other methods in the parent class.
- The `public` and `protected` attributes and methods of `ClubMember` are inherited by `CommitteeMember`. Here we don't need to duplicate the code for `getIdNum()` and `getName()` in the class `CommitteeMember`.

Club Example – The Subclass CommitteeMember

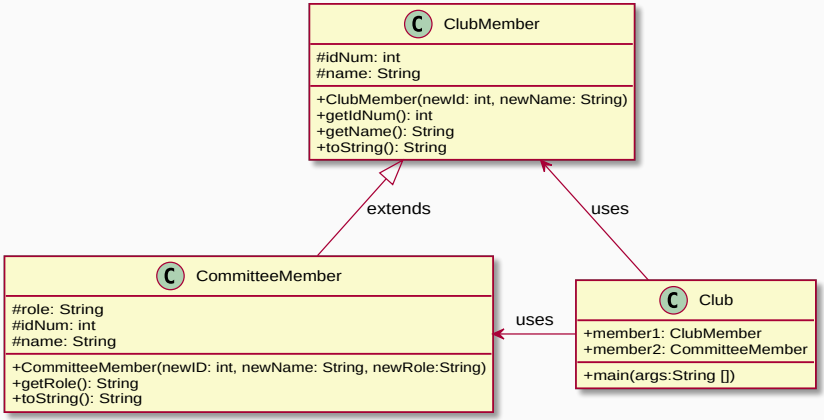
```
1  public class CommitteeMember extends ClubMember {
2
3      /* additional attribute */
4    protected String role;
5
6    /*  Constructor */
7    public CommitteeMember(int newID, String newName, String newRole) {
8       super(newID, newName);
9       role = newRole;
10   }
11
12   /*  getter for new atribute */
13   public String getRole() { return role; }
14
15   /* override toString method*/
16   public String toString() {
17      return("Committee Member Name: " + getName() +
18              "  ID number: " + getIdNum() + " Role: " + getRole());
19   }
20 }
```

Club Example – An Application

```java
/** A short application to test ClubMember and CommitteeMember  */
public class Club {

  public static void main(String[] args) {

    ClubMember member1 = new ClubMember(123, "Fred");
    CommitteeMember member2 = new CommitteeMember(22, "Helen", "Chair");

    System.out.println(member1.toString());
    System.out.println(member2.toString());
  }
}
```

```
$> java Club
Member Name: Fred  ID number: 123
Committee Member Name: Helen  ID number: 22 Role: Chair
```

Club Example – Full UML Class Diagram



Assignments using Subclasses (a first taste of Polymorphism)

Any object of a subclass can be assigned to a variable of its superclass type:

```
1  ClubMember ordMember = new CommitteeMember(325, "James", "Chief");
```

because every “CommitteeMember” is “also a “ClubMember”.

But afterwards, you cannot use subclass-specific methods/attributes!

```
2  System.out.println( ordMember.getRole() );  //  ERROR!!!
3
4  //  However, the line below here is ok.
5  //  What will be the result of this statement?
6  System.out.println( ordMember.toString() );
```

... unless you type cast..

```
2  // this is OK because ordMember refers to a CommitteeMember object!
3  System.out.println((CommitteeMember)ordMember.getRole() );
```

Assignments using Subclasses (cont.)

In general, we cannot assign superclass objects to subclass-typed variables

```
1  CommitteeMember newCtteMember;
2  ClubMember ordMember2 = new ClubMember(123, "Fred");
3  newCtteMember = ordMember2;      //  ERROR!!!
```

```
$ javac ClubUser.java
ClubUser.java:25: error: incompatible types: ClubMember cannot
                                be converted to CommitteeMember
        ctteMember = ordMember2;
                     ^
1 error
```

because not every “ClubMember” is also a “CommitteeMember”.

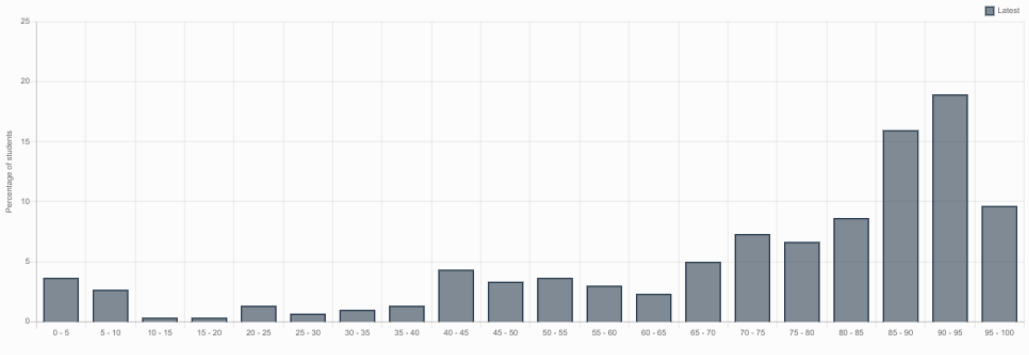
Feedback on A1

Assignment 1 Stats

General statistics

301	3086	72.2 ±26	10.3 ±11.8	1.6 ±1.4
Students	Submissions	Average grade [1]	Average submissions [1]	Average inline feedback entries [1]

Grade statistics

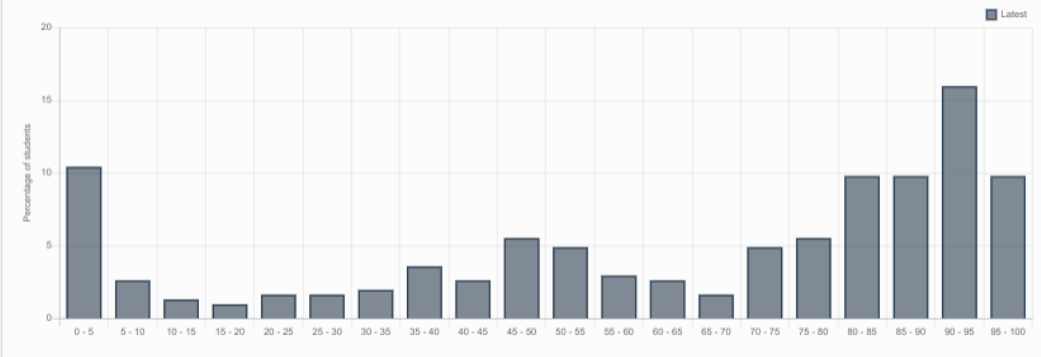


Assignment 1 Stats 2022

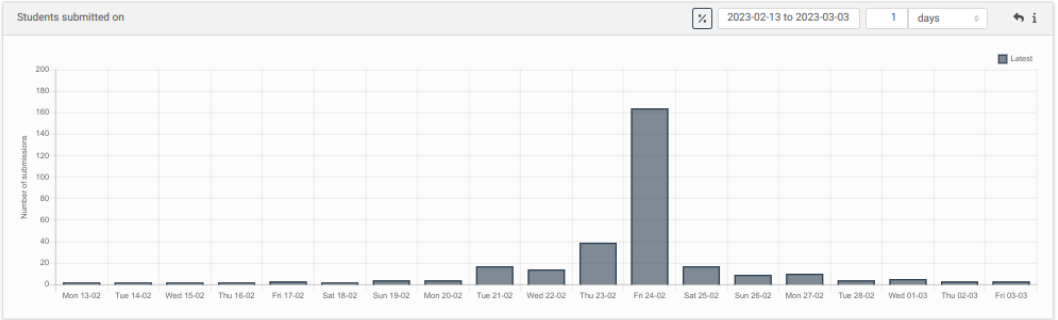
General statistics

307	2462	63.4 ±32	8 ±8.6	2.2 ±2
Students	Submissions	Average grade	Average submissions	Average inline feedback entries

Grade statistics



Assignment 1 Submission Times



Inline feedback

warning: -0.6 penalty, info: just fyi

```
1.  public class Caesar {
2.
3.      public static char rotate(int shift, char c) {
4.          if (Character.isLowerCase(c)) {
5.              return (char) (((c - 'a' + shift) % 26) + 'a');
6.          } else if (Character.isUpperCase(c)) {
7.              return (char) (((c - 'A' + shift) % 26) + 'A');
8.          } else {
9.              return c;
10.         }
11.     }
12.
13.     public static String rotate(int shift, String s) {
14.         String result = "";
15.         for (int i = 0; i < s.length(); i++) {
16.             result += rotate(shift, s.charAt(i));
17.         }
18.         return result;
19.     }
```

Popular Plagiarism Cases: Java Hungry and Chegg.com

Caesar Cipher Program In Java With Output

Caesar cipher technique was founded by Julius caesar. Before looking at the **caesar cipher program in java with output for encryption and decryption**, first, we need to understand the terms plaintext and ciphertext.

Read Also: **Vigenere Cipher Program in Java**

What is plaintext and ciphertext?

plaintext is the input message given by user. In other words, message that needs to be encrypted.

ciphertext is the encrypted message. In other words, message after applying the caesar cipher technique.

What is Caesar cipher?

Caesar cipher is one of the simplest encryption technique. It is also known as the shift cipher, Caesar's cipher, Caesar shift or Caesar's code. Caesar cipher is a type of substitution cipher.

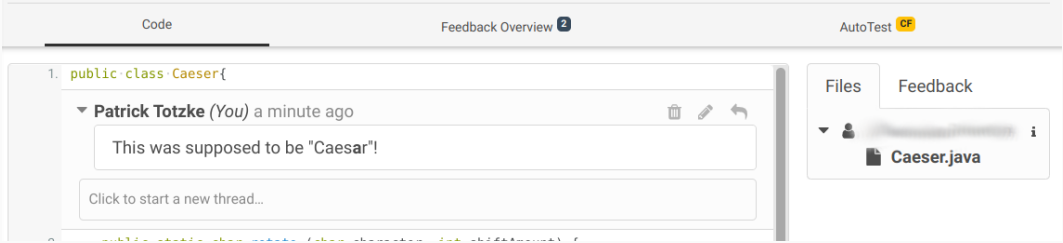
By using this cipher technique we can replace each letter in the plaintext with different one a fixed number of places up or down the alphabet.

```
import java.util.*;
public class CaesarCipherProgram {
   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      System.out.println(" Input the plaintext message : ");
      String plaintext = sc.nextLine();
      System.out.println(" Enter the value by which
      each character in the plaintext
      message gets shifted : ");
      int shift = sc.nextInt();
      String ciphertext = "";
      char alphabet;
      for(int i=0; i < plaintext.length();i++)
      {
          // Shift one character at a time
         alphabet = plaintext.charAt(i);

        // if alphabet lies between a and z
        if(alphabet >= 'a' && alphabet <= 'z')
        {
         // shift alphabet
         alphabet = (char) (alphabet + shift);
         // if shift alphabet greater than 'z'
         if(alphabet > 'z') {
            // reshift to starting position
            alphabet = (char) (alphabet+'a'-'z'-1);
         }
         ciphertext = ciphertext + alphabet;
        }

        // if alphabet lies between 'A'and 'Z'
        else if(alphabet >= 'A' && alphabet <= 'Z') {
         // shift alphabet
```

Infinite Loops!

```
1  while (l < 26){
2      chi_squared_shift[l] = chiSquared(frequency(text, count(text)),
           english);
3      text = rotate(1, text);
4
5      l =+ 1;
6      System.out.println(l);
7  }
```

Caeser


Code
Feedback Overview 2
AutoTest CF
1. public class Caeser{
Patrick Totzke (You) a minute ago
This was supposed to be "Caesar"!
Click to start a new thread...
Files
Feedback
Caeser.java

Mississippi Moon!

```java
public static double[] frequency(String s) {
    double[] freqs = new double[26];
    int[] counts = count(s);
    int total = s.replaceAll("[^a-zA-Z]", "").length();  // <3
    for (int i = 0; i < 26; i++) {
        freqs[i] = (double) counts[i] / total;
    }
    return freqs;
}
```