Role of Programming Languages

T1 Chapter 1

By, Archana Chaudhari

Text Book: T1. T. W. Pratt, M. V. Zelkowitz, "Programming Languages Design and Implementation‖, 4th Ed, PHI, ISBN 81-203-2035-2.

Role of Programming Languages

T1 Chapter 1

Programming Languages

Role of Programming Languages
T1 Chapter 1

- A programming language allows people to create programs that tell machines (Computers) what to do.
- A programming language is a tool for developing executable models for a class of problem domains.
- What makes a Good Language
- Language Paradigms
- Language Standardization
- Internationalization

What makes a Good Language?

- Attributes of a Good Language
  - Clarity, simplicity, and unity
  - Orthogonality
  - Naturalness for the application
  - Support for abstraction
  - Ease of program verification
  - Programming environment
  - Portability of programs
  - Cost of use
    - Cost of Program execution
    - Cost of Program translation
    - Cost of program creation, testing, and use
    - Cost of program maintenance

Clarity, simplicity, and unity

- A Programming language provides a conceptual framework for thinking about the algorithm.
- It should provide a clear, simple and unified set of concepts that can be used as primitives in developing algorithms.
- It should have
    - It has minimum number of different concepts
    - with Rules for their combination being
    - simple and regular.
- This attribute is called **conceptual integrity**
- The syntax of the language should be such that readability of the program can be increased

Orthogonality

- It is one of the most important feature of PL orthogonality is the property that means **" Changing A does not change B".**
- If I take Real world example of an orthogonal system would be a **radio**, where changing the station does not change the volume and vice versa.
- When the features of a language are orthogonal, language is **easier to learn** and **programs are easier to write** because only few exceptions and special cases to be remembered.
- For instance-pointer should be able to point to any type of variable or data structure

Naturalness for the application

- The syntax of a PL should be such that it should follow logical structure of the algorithm
- program structure reflects the logical structure of algorithm
- Various algorithms such as sequential, concurrent, logic algorithm and so on have different natural structures which should be represented by the programming language.
- The language should provide appropriate data structures, operations and control structures for the problem to be solved

Support for abstraction

- Abstraction means hiding the implementation details
- By this attribute programmer can concentrate only on abstract properties without bothering for their implementation details
- ADA and C++ are the languages support the abstraction feature

Ease of program verification

- **Reusability:**The reusability of program written in a language is always a central concern.
- A program is checked by various testing technique like **Formal verification method: Desk checking, Input output test checking.**
- We verify the program by many more techniques. A language that makes program verification difficult maybe far more troublesome to use.
- **Simplicity of semantic and syntactic structure is a primary aspect that tends to simplify program verification.**

Programming environment

- An appropriate programming environment adds an extra utility and make language to be implemented easily like
- **The availability of**- Reliable- Efficient - Well documentation
- **Speeding up creation and testing by-special Editors- testing packages**
- **Facility- Maintaining and Modifying- Multi Version of program software product.**

Portability of programs

- Programming language should be portable means it should be easy to transfer a program from which they are developed to the other computer.
- A program whose definition is independent of features of a Particular machine forms can only support **Portability. Example: Ada, FORTRAN, C, c++, Java.**

Cost of use

a) Cost of program execution

- Large production program that will be executed repeatedly
- The program execution cost greatly reduced due to use of optimizing compilers, efficient register allocation and design of efficient runtime support mechanism

b) Cost of Program translation

- For compiling the large program , compiler takes too much time it’s increase overall cost
- It is import to have fast and efficient compiler rather than a compiler that produce optimized executable code

Cost of use

a) Cost of program creation, testing, and use
   - Involved in program designing, coding, testing and modifying
   - The smalltalk and perl are cost effective language

b) Cost of program maintenance
   - The maintenance cost can be 4 times more than of development cost.
   - It is dependent upon readability

Language Paradigm

- Imperative or procedural Languages
- Applicative or Functional Languages
- Rule-based or Logical Languages
- Object-oriented programming

Imperative or procedural Languages

- These are command driven or statement oriented language.
- The basic concept is the machine state, the set of all values for all memory location in the computer.
- A prog. consists of sequence of statement and the execution of each statement causes the computer to change the value of one or more location in its memory
- Syntax:

  Statement 1

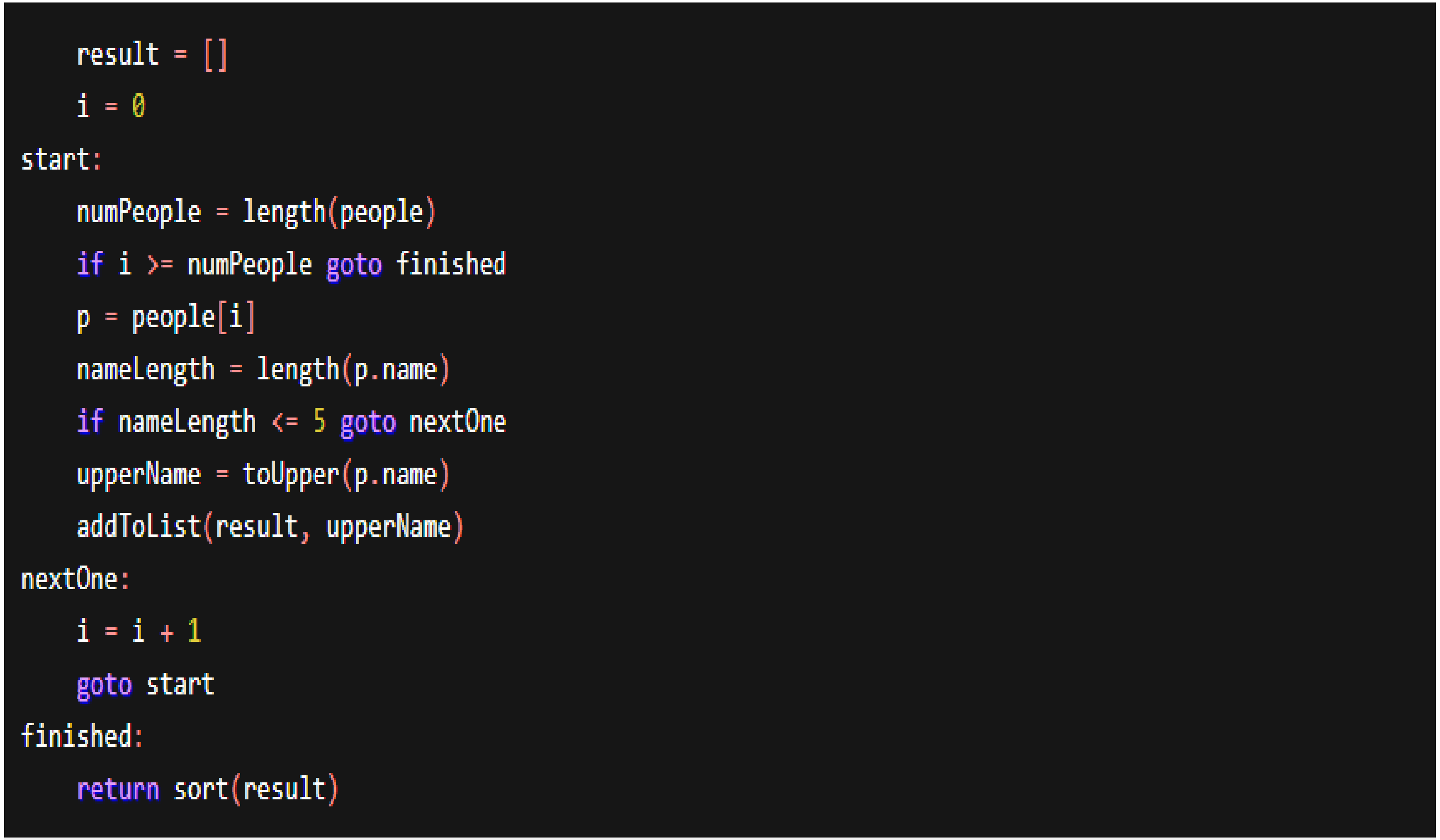
  Statement 2

    - -
    -

  Statement n

- Eg. FORTRAN, C

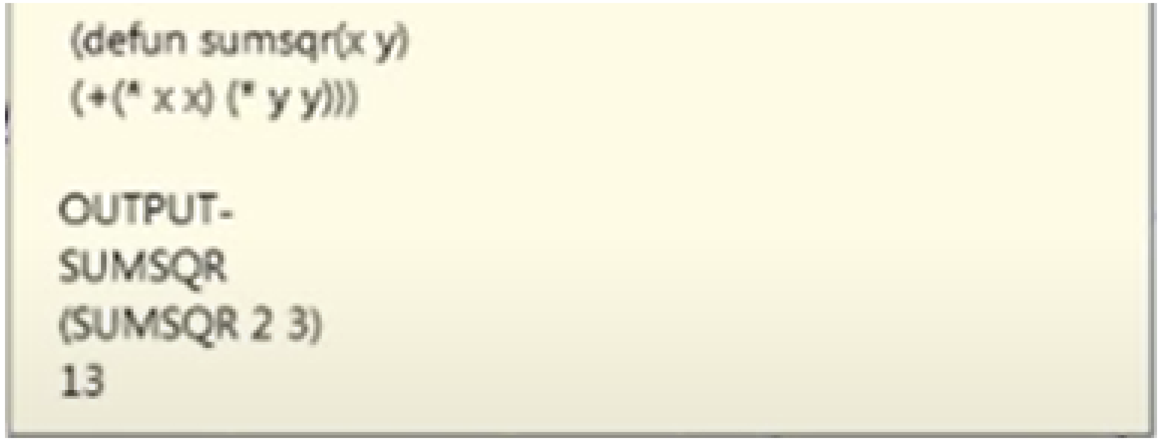
Example Imperative Language



```
    result = []
    i = 0
start:
    numPeople = length(people)
    if i >= numPeople goto finished
    p = people[i]
    nameLength = length(p.name)
    if nameLength <= 5 goto nextOne
    upperName = toUpper(p.name)
    addToList(result, upperName)
nextOne:
    i = i + 1
    goto start
finished:
    return sort(result)
```

Applicative or Functional Languages

- Function is the basic building blocks
- Programming with function calls that avoid any global state.
- In **functional programming**, control flow is expressed by combining function calls, rather than by assigning values to variables:
- Function$_n$(--- Function$_2$(Function1(data))
- Eg. LISP



```
(defun sumsqr(x y)
(+(* x x) (* y y)))

OUTPUT-
SUMSQR
(SUMSQR 2 3)
13
```

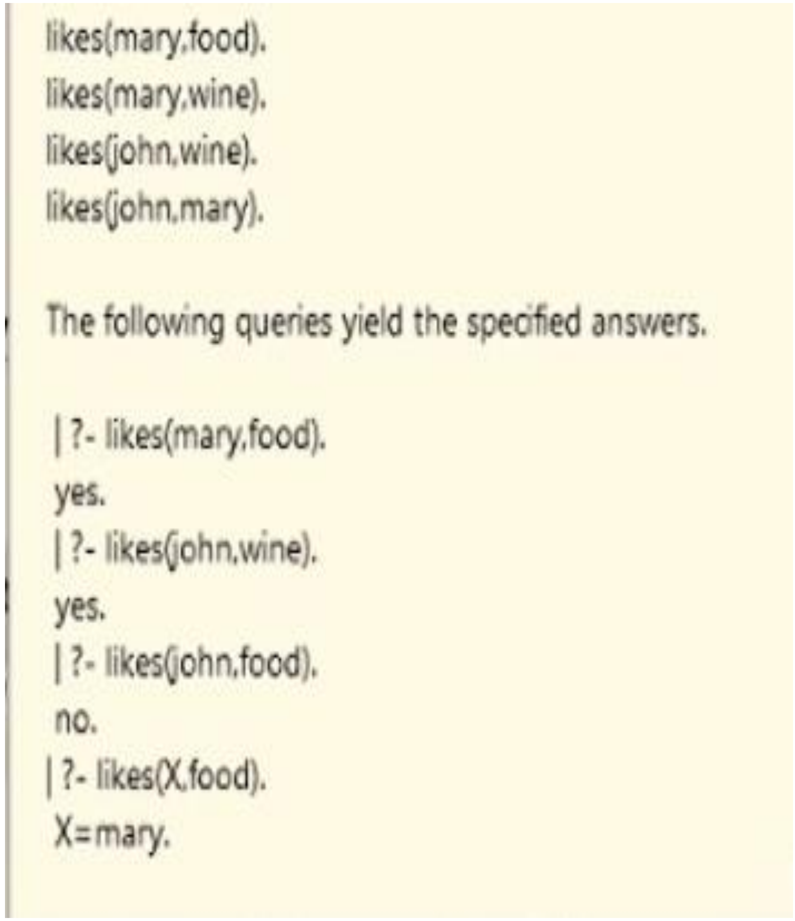
Rule-based or Logical Languages

- Programming by specifying a set of facts and rules.
- A rule based program is a collection of declaration which are true about the desired result, these are called facts
- Syntax:

  enabling cond1->action1

  ……

  Enabling condn->actionn
- Unification and backtracking to find solutions (i.e.. satisfy goals) takes place automatically.
- Eg. PROLOG



```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).
```

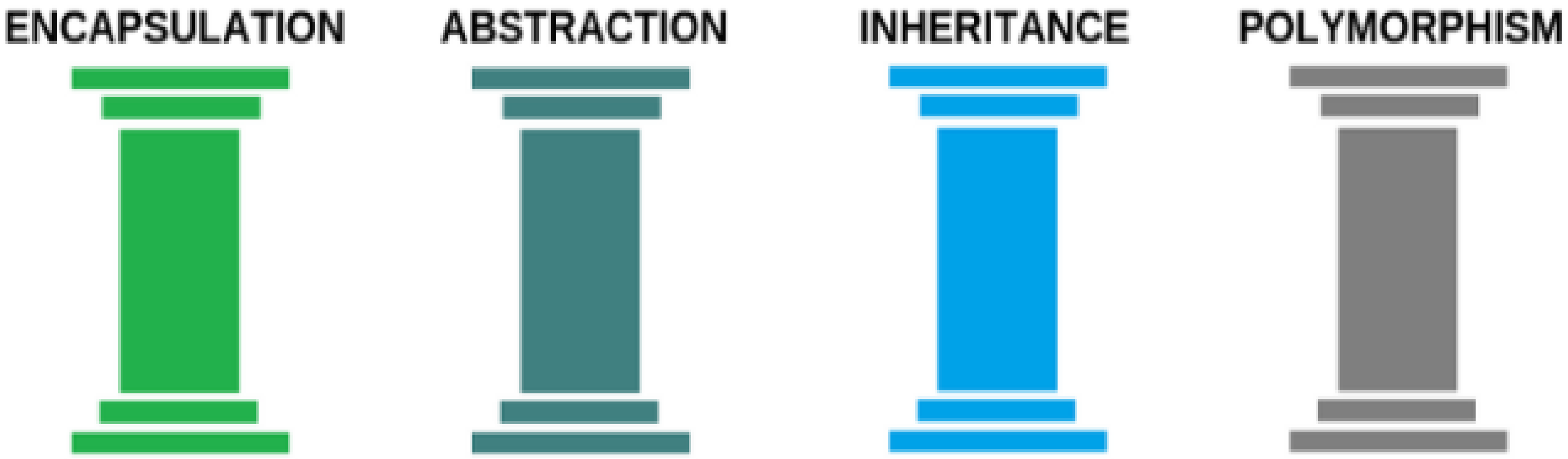
The following queries yield the specified answers.

```
| ?- likes(mary,food).
yes.
| ?- likes(john,wine).
yes.
| ?- likes(john,food).
no.
| ?- likes(X,food).
X=mary.
```

Object-oriented programming

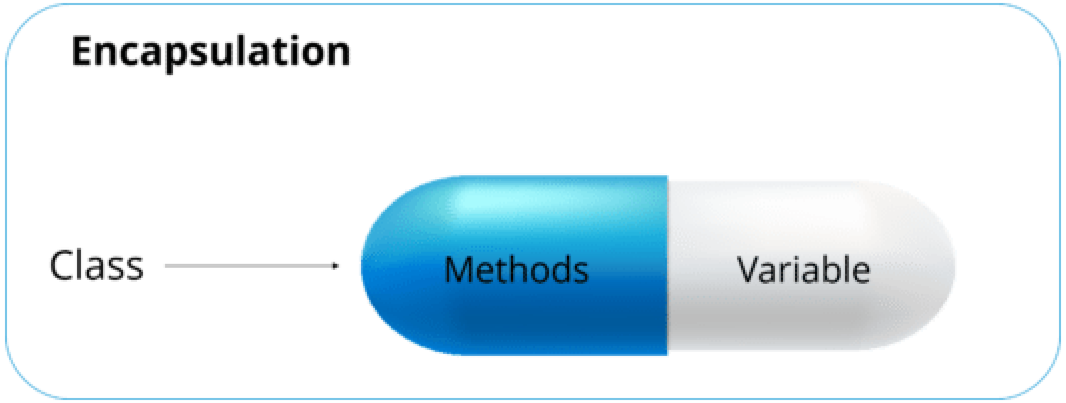
- OOP as a programming language model built around “objects”, which can contain data in the form of fields (often known as attributes), and code, in the form of procedures (often known as methods).
- An object can be defined as an instance of a class, and there can be multiple instances of a class in a program.
- objects are the most important part of our program.

4 pillars of OOP



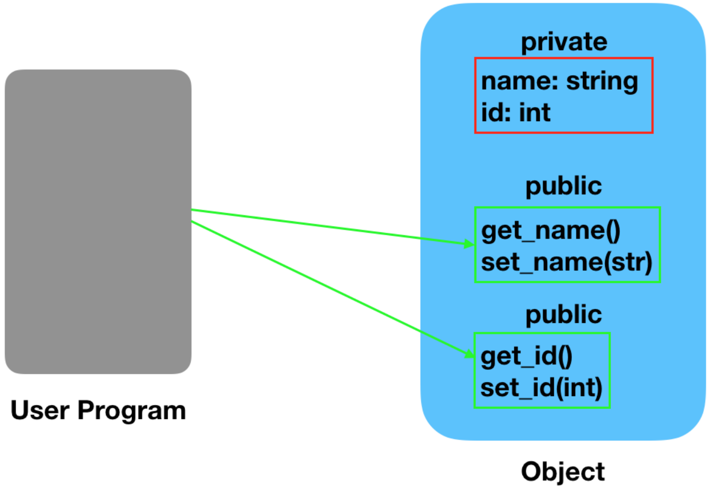
Encapsulation

- Encapsulation means wrapping up data and member function (Method) together into a single unit i.e. class.
- Encapsulation automatically achieve the concept of data hiding providing security to data by making the variable as private and expose the property to access the private data which would be public.
- Encapsulation helps in protecting our data while binding the data variables and functions into one unit.



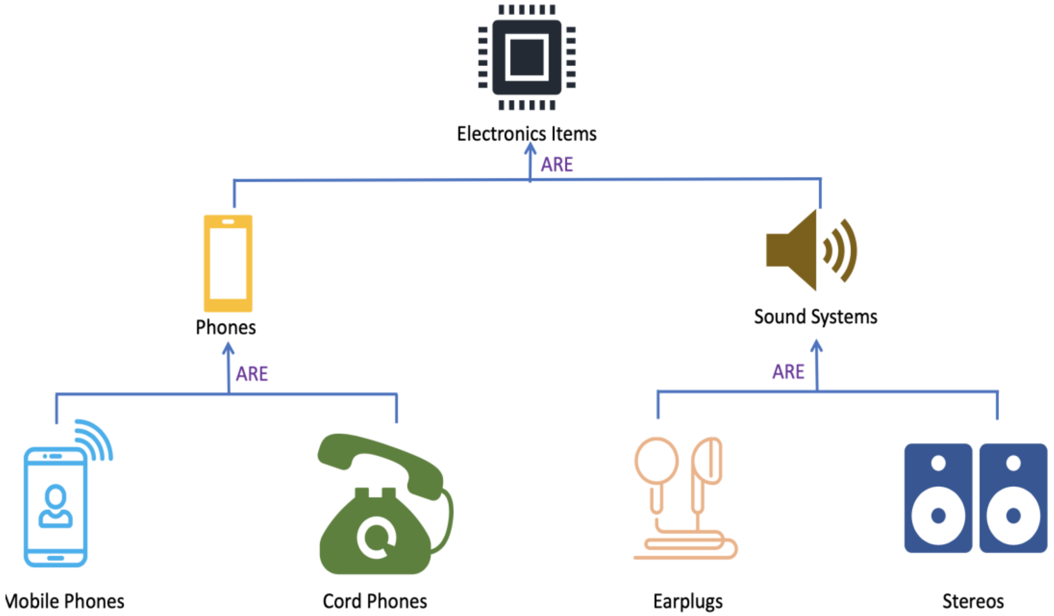
Abstraction

- Abstraction is the process of showing only essential/necessary features of an entity/object to the outside world and hide the other irrelevant information.
- For example to ON your TV we only have a power button, It is not required to understand how infra-red waves are getting generated in TV remote control.



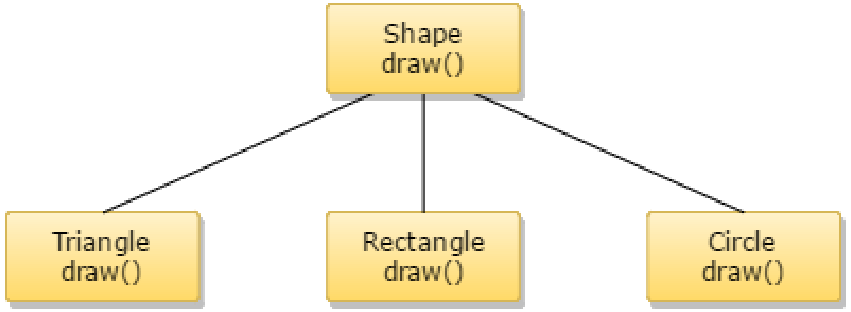
Inheritance

- The ability of creating a new class from an existing class.
- Inheritance is when an object acquires the property of another object.
- Inheritance allows a class (subclass) to acquire the properties and behavior of another class (super-class).
- It helps to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time.



Polymorphism

- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means "many forms".
- A subclass can define its own unique behavior and still share the same functionalities or behavior of its parent/base class.
- A subclass can have their own behavior and share some of its behavior from its parent class not the other way around. A parent class cannot have the behavior of its subclass.



Polymorphism

Language Standardization

- Proprietary Standards: These are definition by the company that developed and owns the language
- Consensus Standards: Are the major methods to ensure uniformity among several implementation of a language
- To use standards effectively we need to address 3 issues:
  - Timeliness: when do we standardize a language?
  - Conformance: what does it mean for a program to adhere to a standard and for a compiler to compile a standard?
  - Obsolescence: when does a standard age, and how does it get modified?

Internationalization

With the globalization of commerce and the WWW, programming is increasingly a global activity, and it is important for lang. to be readily useable in multiple countries.

Some of the relevant issues are as follows:

- Collating Sequences: In what collating sequence should the characters be ordered?
- Country-specific date formats: 11/26/21 in US, 26/11/21 in England, 26.11.21 in france
- Country-specific time formats: 5:40pm in US is 17:40 in Japan
- Time Zones
- Currency: representation of currency varies by country

Thank You