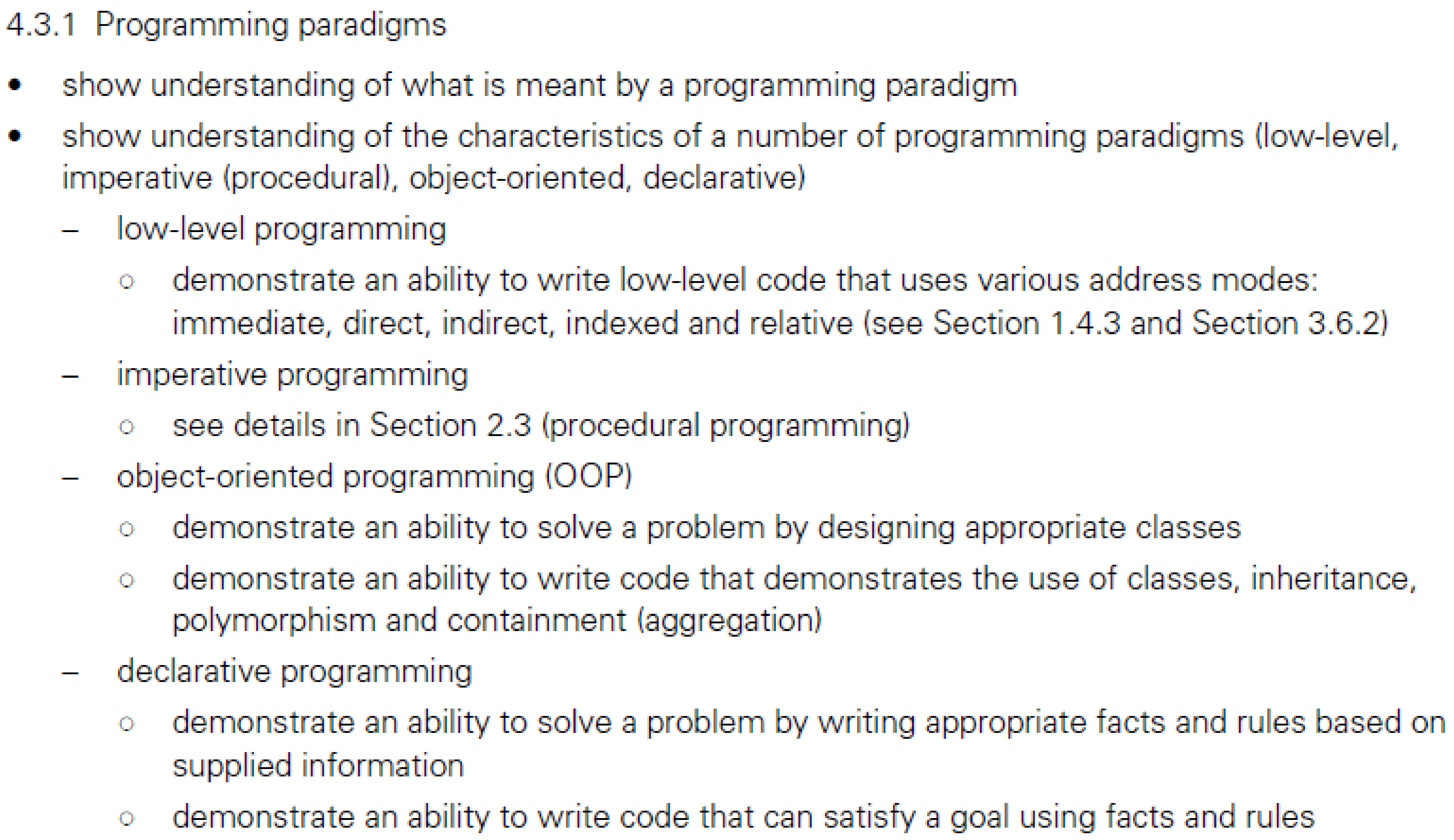
Chapter 27
Object-oriented Programming (OOP)

Learning objectives

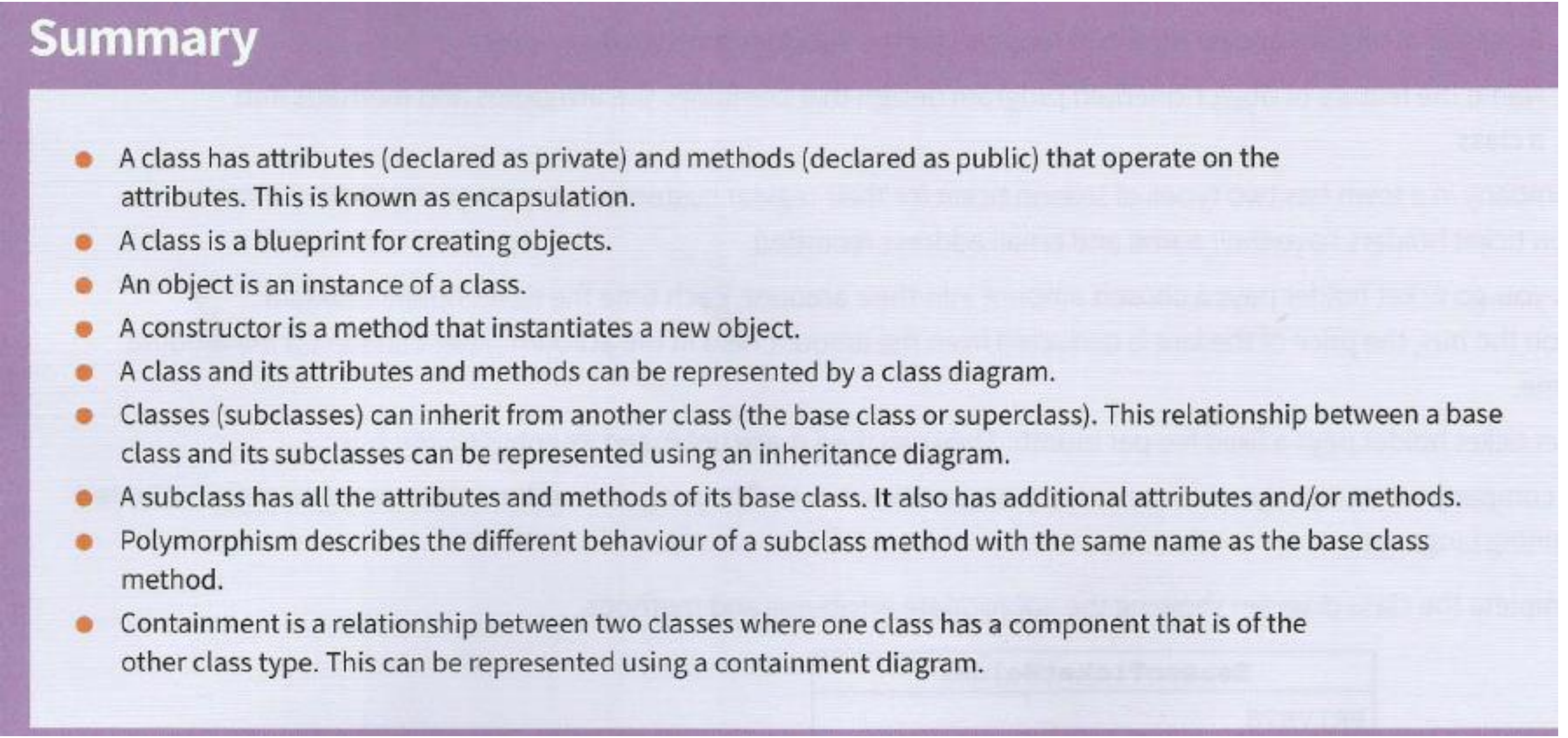
By the end of this chapter you should be able to:

- solve a problem by designing appropriate classes
- write code that demonstrates the use of classes, inheritance, polymorphism and containment (aggregation).



4.3.1 Programming paradigms

- show understanding of what is meant by a programming paradigm
- show understanding of the characteristics of a number of programming paradigms (low-level, imperative (procedural), object-oriented, declarative)
  - – low-level programming
    - ○ demonstrate an ability to write low-level code that uses various address modes: immediate, direct, indirect, indexed and relative (see Section 1.4.3 and Section 3.6.2)
  - – imperative programming
    - ○ see details in Section 2.3 (procedural programming)
  - – object-oriented programming (OOP)
    - ○ demonstrate an ability to solve a problem by designing appropriate classes
    - ○ demonstrate an ability to write code that demonstrates the use of classes, inheritance, polymorphism and containment (aggregation)
  - – declarative programming
    - ○ demonstrate an ability to solve a problem by writing appropriate facts and rules based on supplied information
    - ○ demonstrate an ability to write code that can satisfy a goal using facts and rules



Summary

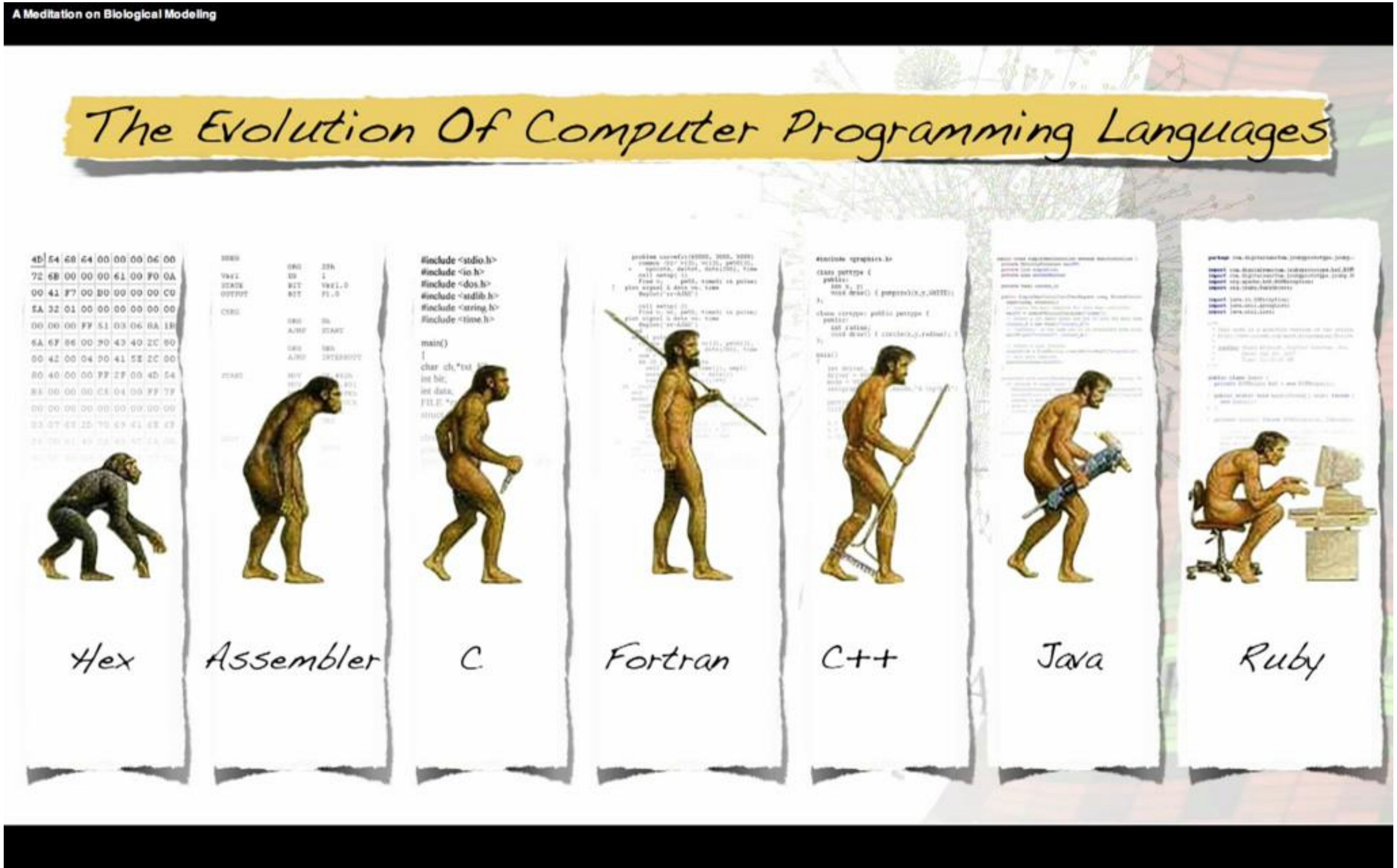
- A class has attributes (declared as private) and methods (declared as public) that operate on the attributes. This is known as encapsulation.
- A class is a blueprint for creating objects.
- An object is an instance of a class.
- A constructor is a method that instantiates a new object.
- A class and its attributes and methods can be represented by a class diagram.
- Classes (subclasses) can inherit from another class (the base class or superclass). This relationship between a base class and its subclasses can be represented using an inheritance diagram.
- A subclass has all the attributes and methods of its base class. It also has additional attributes and/or methods.
- Polymorphism describes the different behaviour of a subclass method with the same name as the base class method.
- Containment is a relationship between two classes where one class has a component that is of the other class type. This can be represented using a containment diagram.

How to test?

- https://papers.xtremepape.rs/CAIE/AS%20and%20A%20Level/Computer%20Science%20(9608)/9608_s18_qp_42.pdf

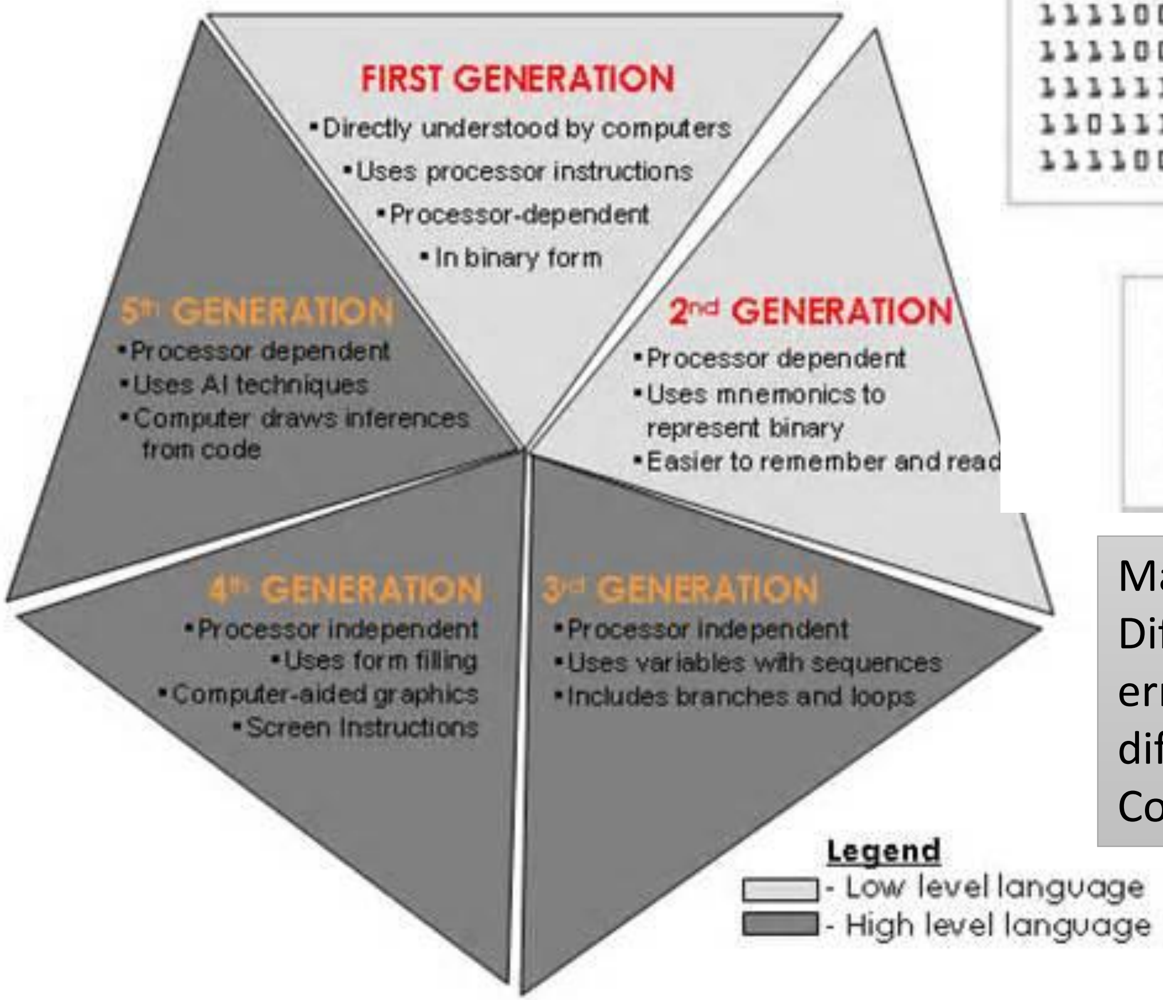
- https://papers.xtremepape.rs/CAIE/AS%20and%20A%20Level/Computer%20Science%20(9608)/9608_s19_qp_41.pdf

Characteristics of programming paradigms


A Meditation on Biological Modeling
The Evolution Of Computer Programming Languages
Hex
Assembler
C
Fortran
C++
Java
Ruby

Characteristics of programming paradigms

Low-level paradigm: machine code



FIRST GENERATION
- Directly understood by computers
- Uses processor instructions
- Processor-dependent
- In binary form

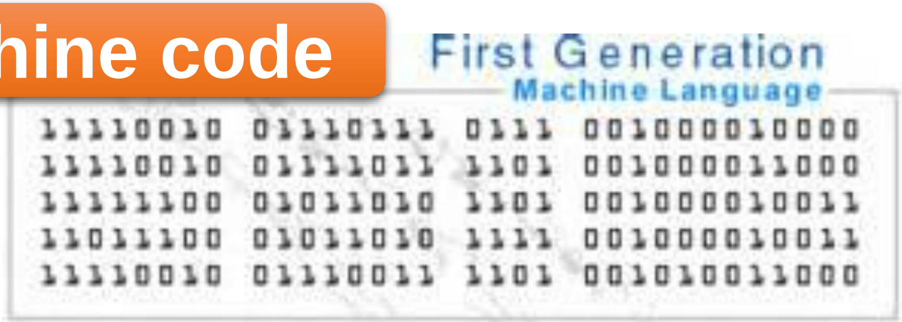
2nd GENERATION
- Processor dependent
- Uses mnemonics to represent binary
- Easier to remember and read

3rd GENERATION
- Processor independent
- Uses variables with sequences
- Includes branches and loops

4th GENERATION
- Processor independent
- Uses form filling
- Computer-aided graphics
- Screen Instructions

5th GENERATION
- Processor dependent
- Uses AI techniques
- Computer draws inferences from code

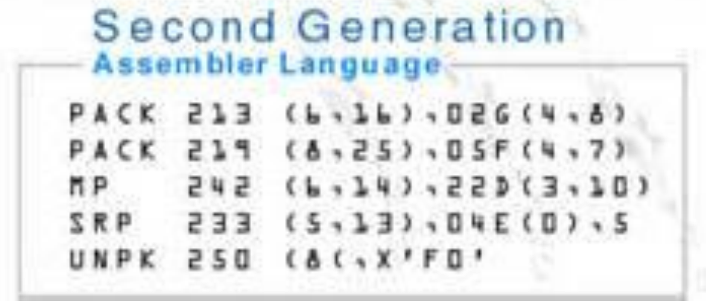
Legend
- ▭ - Low level language
- ▬ - High level language



First Generation

Machine Language

```
11110010 01110111 0111 001000010000
11110010 01111011 1101 001000011000
11111100 01011010 1101 001000010011
11011100 01011010 1111 001000010011
11110010 01110011 1101 001010011000
```



Second Generation

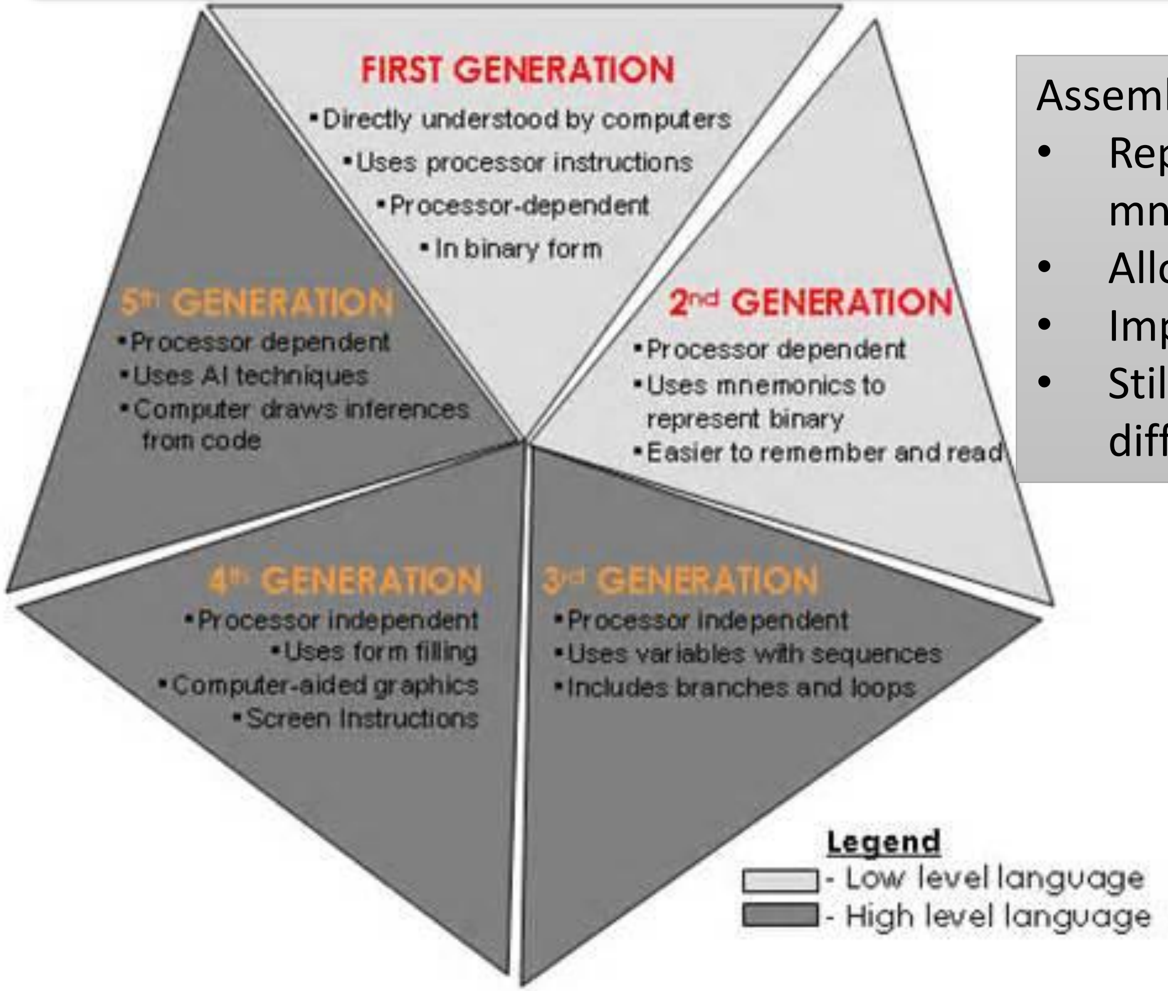
Assembler Language

```
PACK 213 (6,16),026(4,8)
PACK 219 (8,25),05F(4,7)
MP   242 (6,14),22D(3,10)
SRP  233 (5,13),04E(0),5
UNPK 250 (8(,X'F0'
```

Machine code (1 GL):
Difficult to write and lead to many errors in program code that were difficult to find
Communicate in binary

Characteristics of programming paradigms

Low-level paradigm: assembly language



FIRST GENERATION
- Directly understood by computers
- Uses processor instructions
- Processor-dependent
- In binary form

2nd GENERATION
- Processor dependent
- Uses mnemonics to represent binary
- Easier to remember and read

3rd GENERATION
- Processor independent
- Uses variables with sequences
- Includes branches and loops

4th GENERATION
- Processor independent
- Uses form filling
- Computer-aided graphics
- Screen Instructions

5th GENERATION
- Processor dependent
- Uses AI techniques
- Computer draws inferences from code

Legend

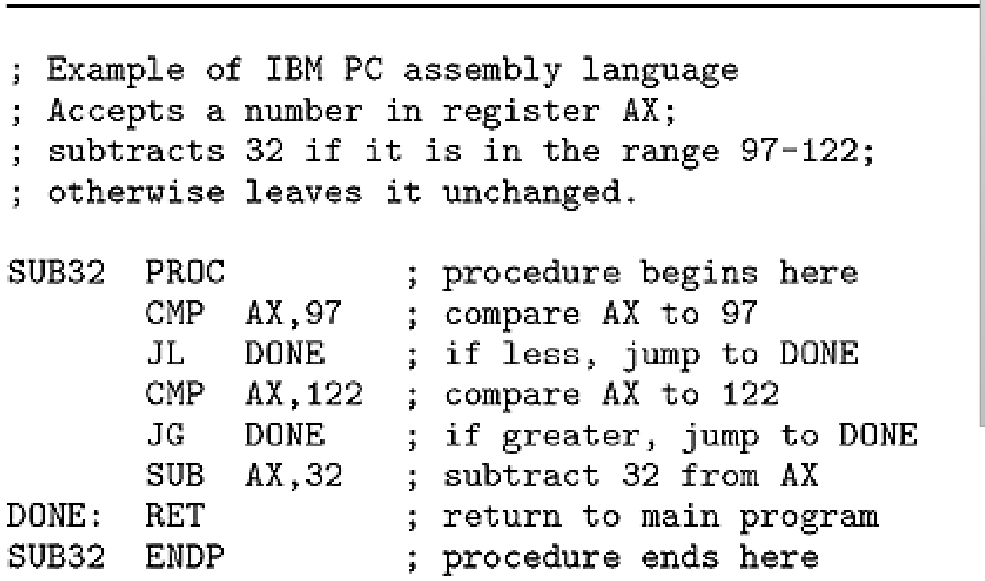
▢ - Low level language

■ - High level language

Assembly languages(2 GL):

- Replace machine code operations with mnemonics
- Allow labels for memory address
- Improvement over machine code
- Still prone to errors and code still difficult to debug, correct and maintain

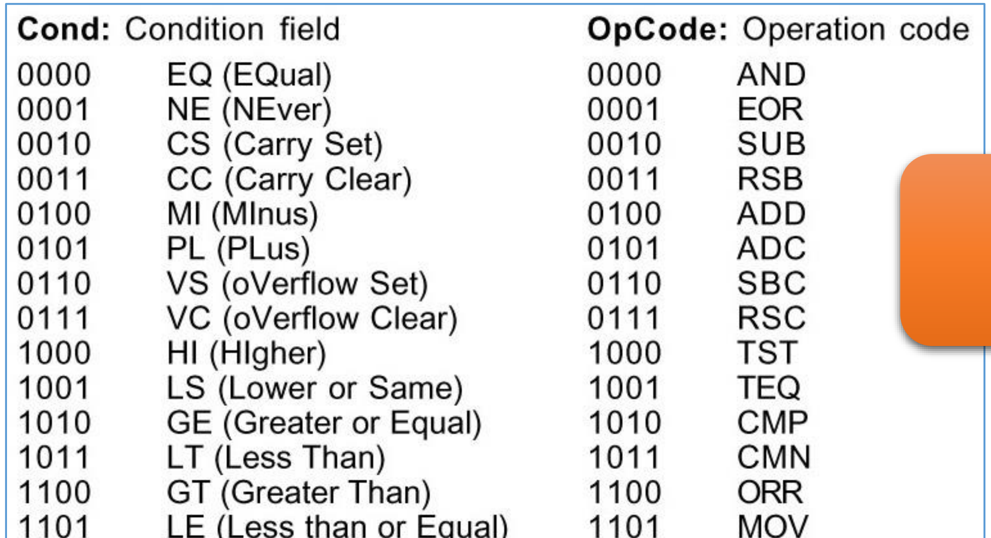
Characteristics of programming paradigms



```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32  PROC         ; procedure begins here
       CMP  AX,97   ; compare AX to 97
       JL   DONE    ; if less, jump to DONE
       CMP  AX,122  ; compare AX to 122
       JG   DONE    ; if greater, jump to DONE
       SUB  AX,32   ; subtract 32 from AX
DONE:  RET          ; return to main program
SUB32  ENDP         ; procedure ends here
```

Assembly languages(2 GL):

- Replace machine code operations with mnemonics
- Allow labels for memory address
- Improvement over machine code
- Still prone to errors and code still difficult to debug, correct and maintain



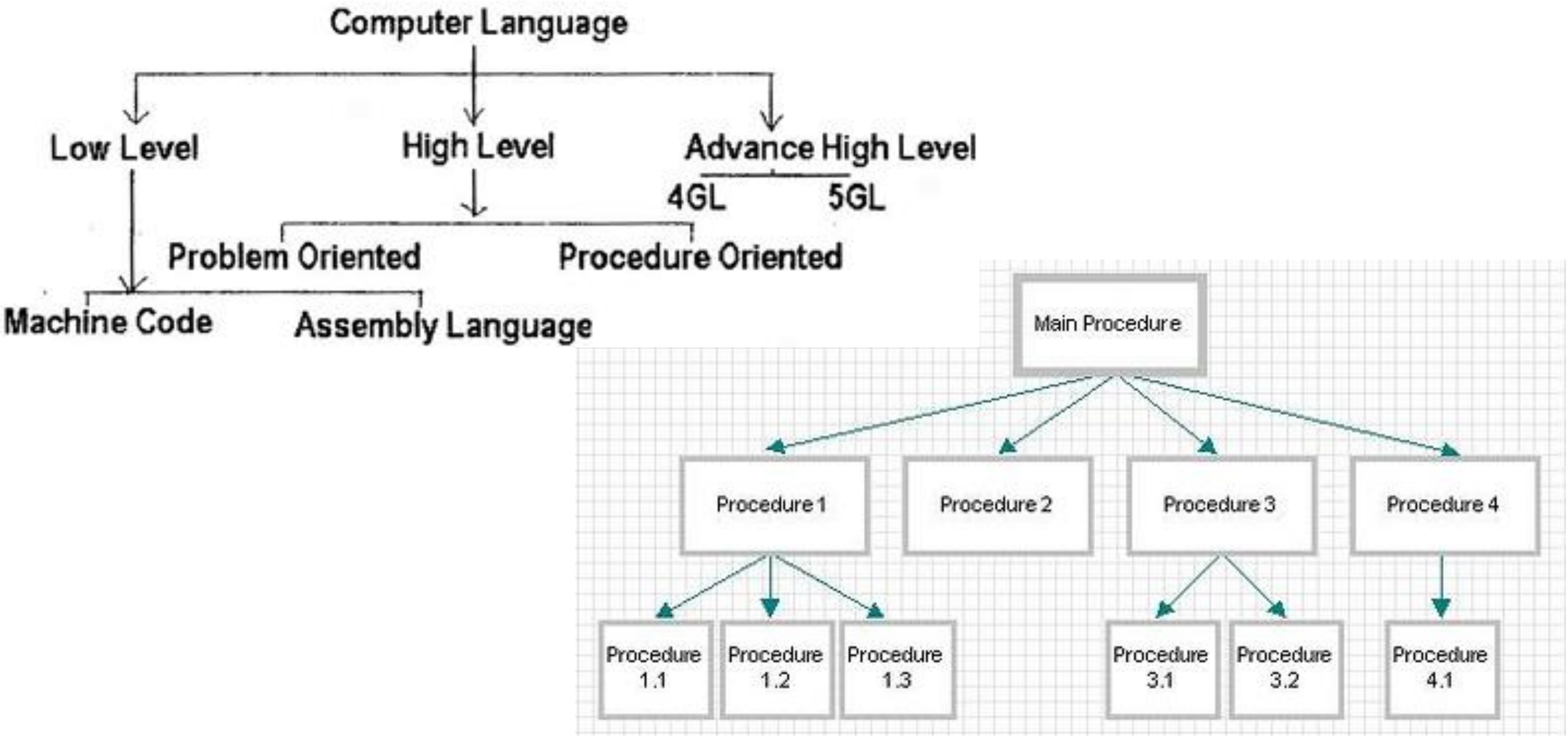
Cond: Condition field		**OpCode:** Operation code	
0000	EQ (EQual)	0000	AND
0001	NE (NEver)	0001	EOR
0010	CS (Carry Set)	0010	SUB
0011	CC (Carry Clear)	0011	RSB
0100	MI (MInus)	0100	ADD
0101	PL (PLus)	0101	ADC
0110	VS (oVerflow Set)	0110	SBC
0111	VC (oVerflow Clear)	0111	RSC
1000	HI (HIgher)	1000	TST
1001	LS (Lower or Same)	1001	TEQ
1010	GE (Greater or Equal)	1010	CMP
1011	LT (Less Than)	1011	CMN
1100	GT (Greater Than)	1100	ORR
1101	LE (Less than or Equal)	1101	MOV

Low-level paradigm: assembly language

Characteristics of programming paradigms

Procedural paradigm

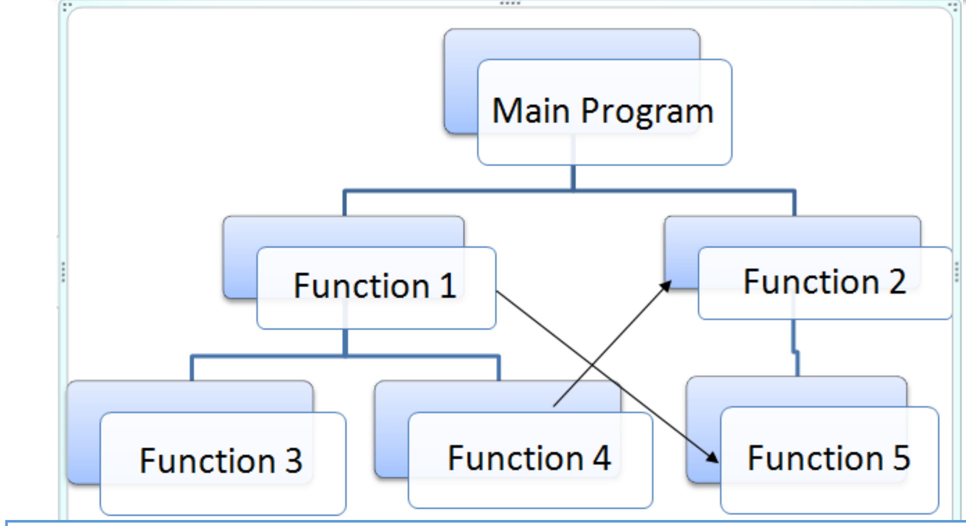
- Third-generation programming language
- Or High level programming language



Characteristics of programming paradigms

Procedural paradigm

- Third-generation programming language
- Or High level programming language



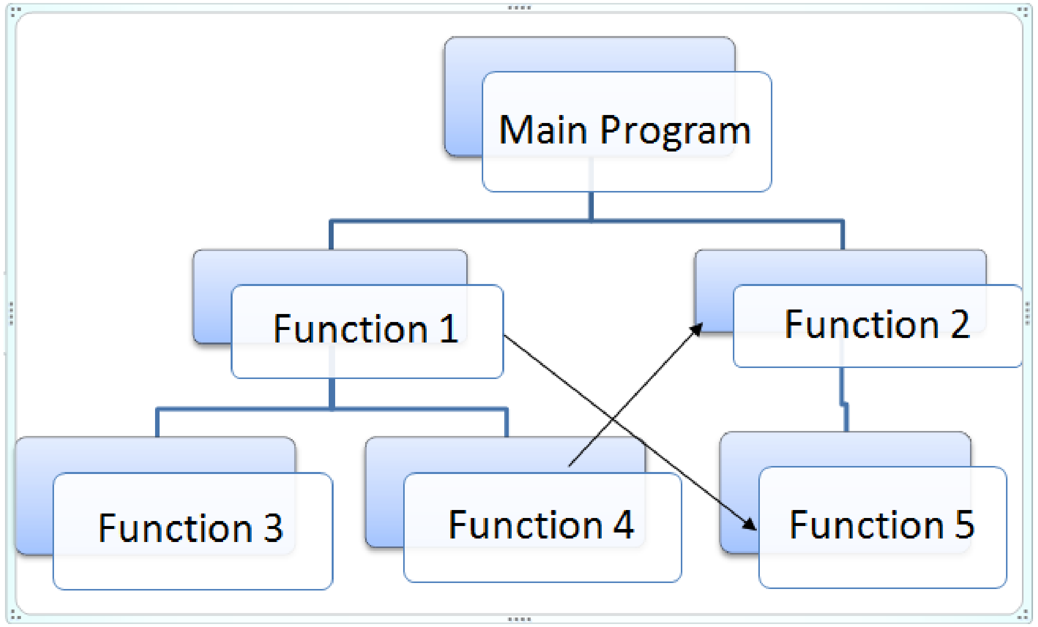
The word "procedure" is the key element here to notice. It means "a set of procedures" which is a "set of subroutines" or a "set of functions".

In a POP method, emphasis is given to functions or subroutines. Functions are a set of instructions which performs a particular task. Functions are called repeatedly in a program to execute tasks performed by them. For example, a program may involve collecting data from user (reading), performing some kind of calculations on the collected data (calculation), and finally displaying the result to the user when requested (printing). All the 3 tasks of reading, calculating and printing can be written in a program with the help of 3 different functions which performs these 3 different tasks.

Characteristics of programming paradigms

Procedural paradigm

- Third-generation programming language
- Or High level programming language



The word “procedure” is the key element here to notice. It means “a set of procedures” which is a “set of subroutines” or a “set of functions“.

In POP method, a problem is viewed as a sequence of tasks to be implemented like reading, performing calculations, displaying results etc. All the tasks are analysed first and later functions/procedures are developed to implement all these tasks in a program.

Characteristics of programming paradigms

Procedural paradigm

- Third-generation programming language
- Or High level programming language

Problem-oriented: they use a language and syntax appropriate to the type of problem being solved.

FORTRAN (FORmula Translation)
ALGOL(ALGOrithmic Language)
BASIC (Beginners All purpose Symbolic Instruction Code)

} Scientific and engineering problems

→ Teaching

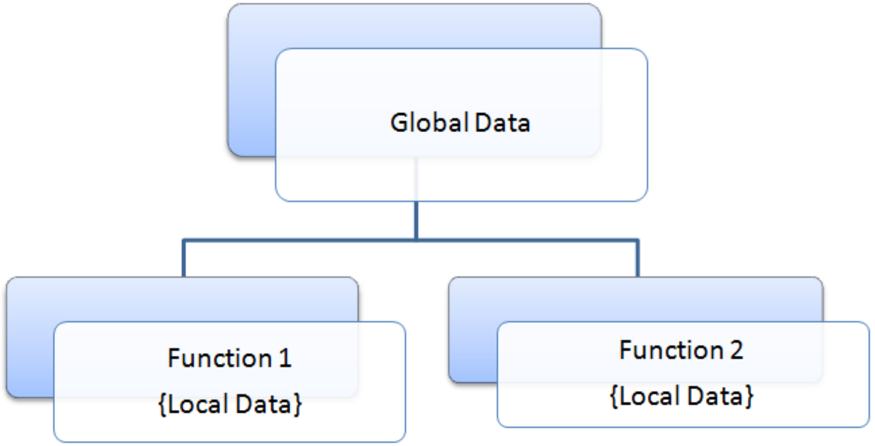
Visual Basic.Net → Sit inside the Microsoft .Net Framework and allow for development of Windows-based applications

Characteristics of programming paradigms

Procedural paradigm

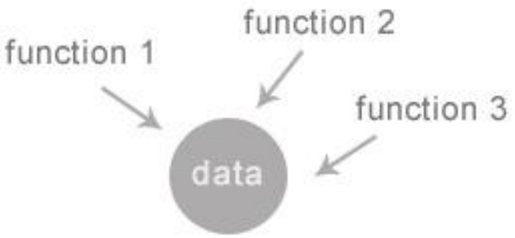
- Third-generation programming language
- Or High level programming language

Object-oriented paradigm



Problem with POP:

- Handling of data: Many functions can modif block of data
- notable drawbacks in creating reusable software components:



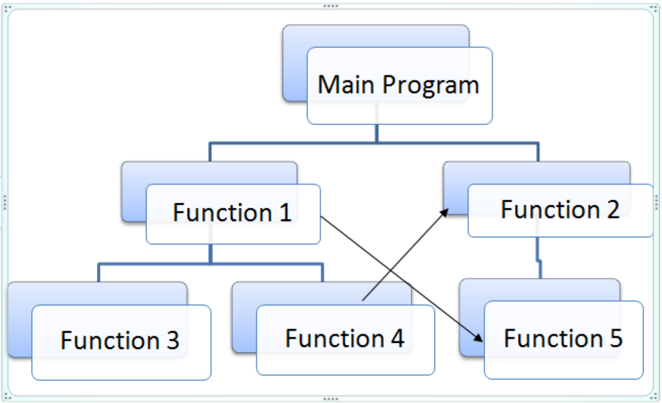
separate the data structures and algorithms/function

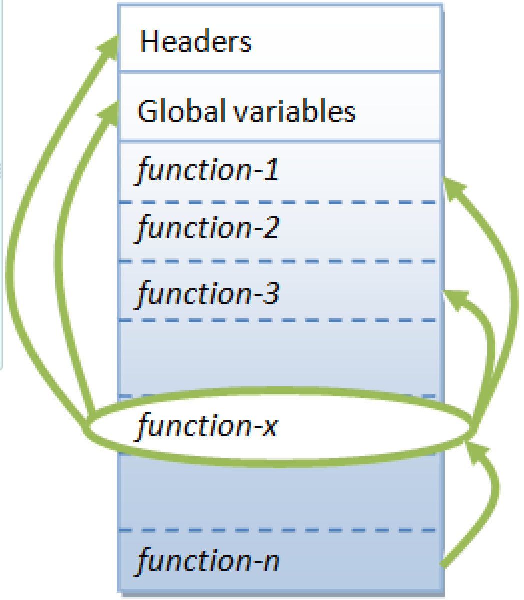
Characteristics of programming paradigms

Procedural paradigm

- Third-generation programming language
- Or High level programming language

Object-oriented paradigm





A function (in C) is not well-encapsulated

Problem with POP:

- Handling of data: Many functions can modify a given block of data
- notable drawbacks in creating reusable software components:

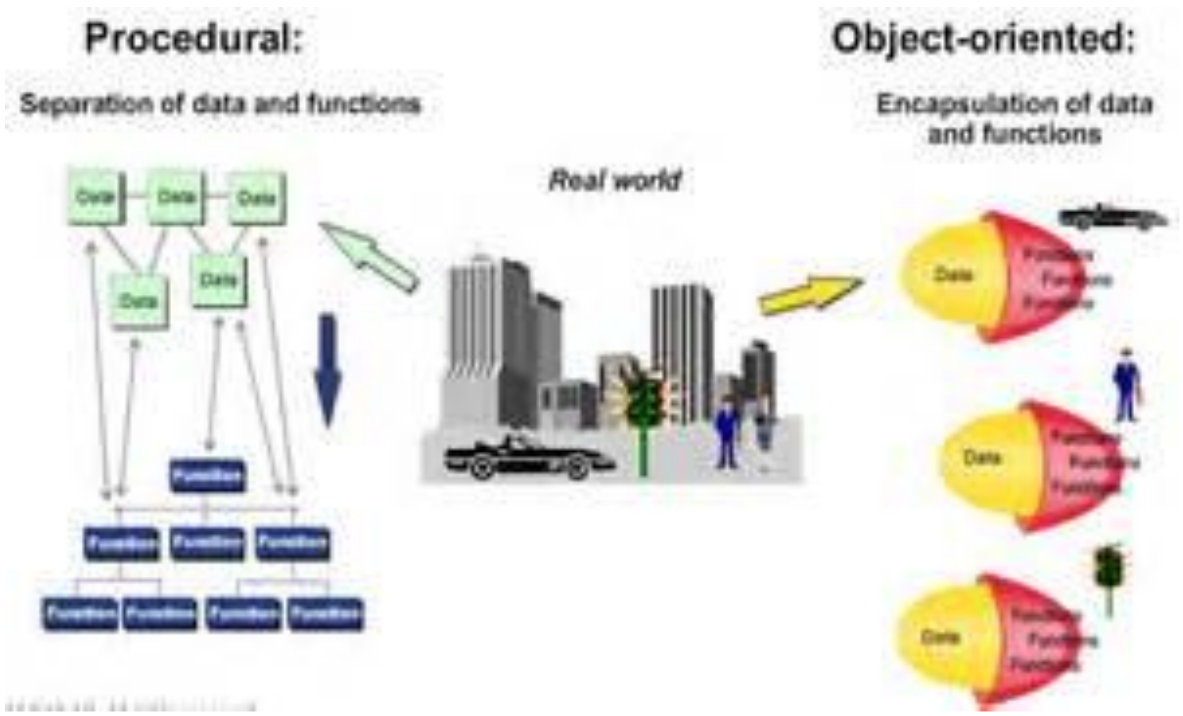
separate the data structures and algorithms/function

Characteristics of programming paradigms

Procedural paradigm

- Third-generation programming language
- Or High level programming language

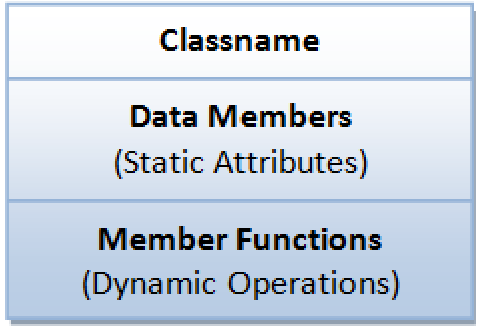
Object-oriented paradigm



Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language

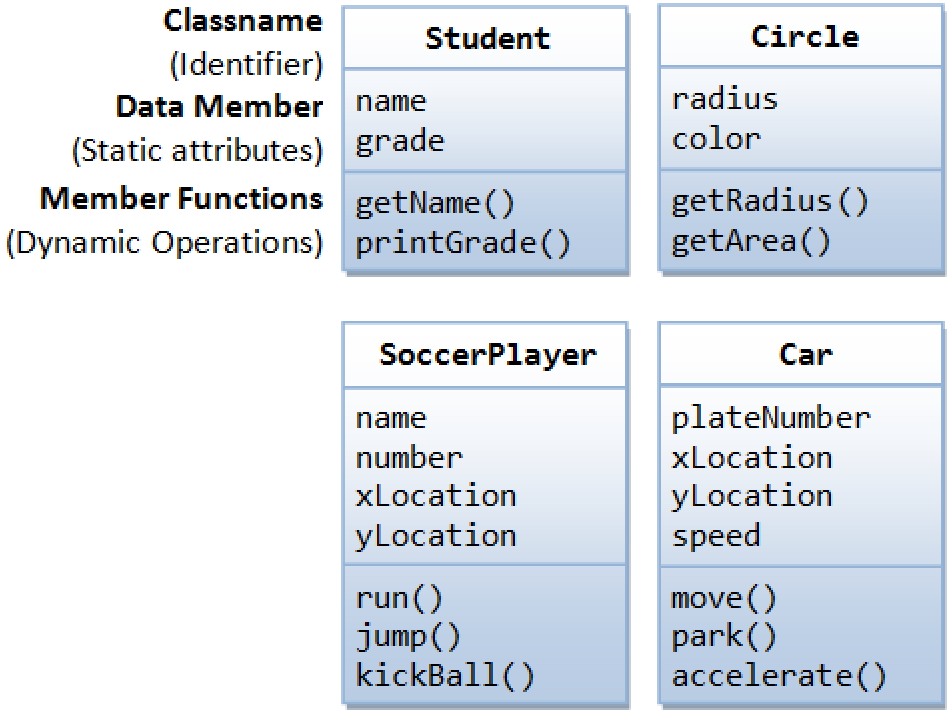


Classname

Data Members
(Static Attributes)

Member Functions
(Dynamic Operations)

A class is a 3-compartment box encapsulating data and functions



Classname
(Identifier)
Data Member
(Static attributes)
Member Functions
(Dynamic Operations)

Student
name
grade
getName()
printGrade()

Circle
radius
color
getRadius()
getArea()

SoccerPlayer
name
number
xLocation
yLocation
run()
jump()
kickBall()

Car
plateNumber
xLocation
yLocation
speed
move()
park()
accelerate()

Examples of classes

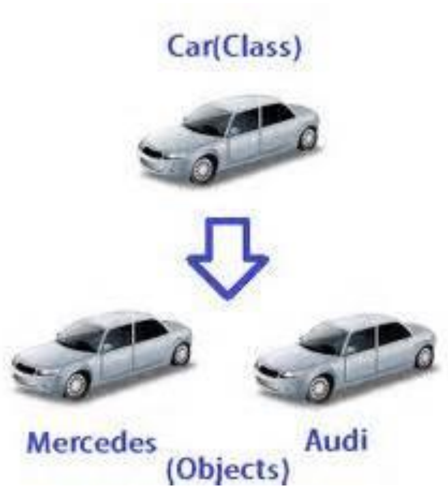
http://evinw.com/oop/

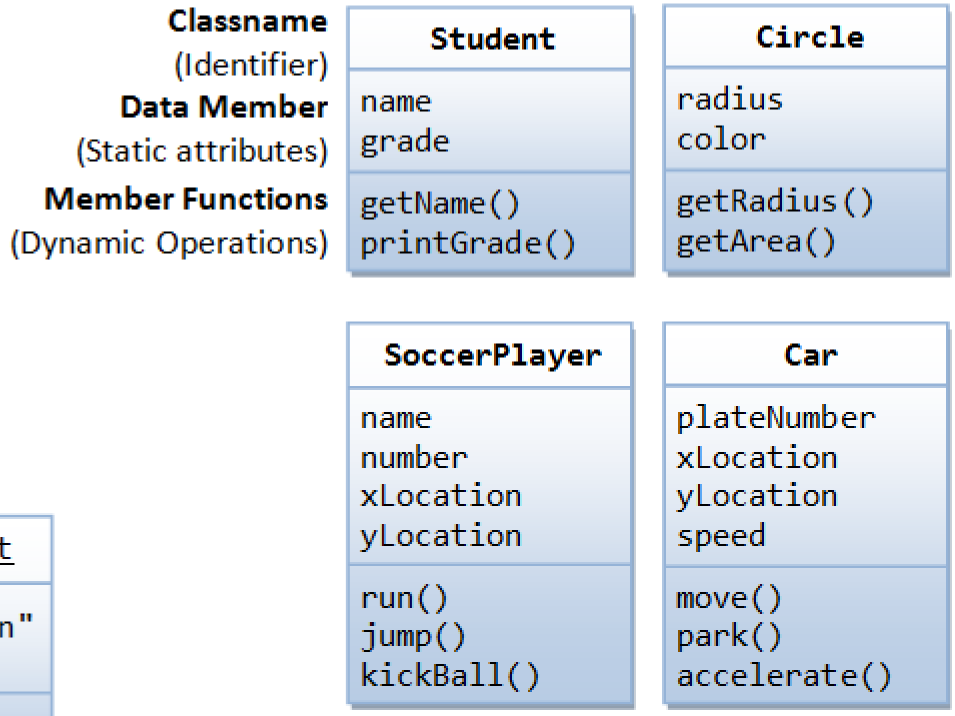
https
amming/cpp/cp3_OOP.html

Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language





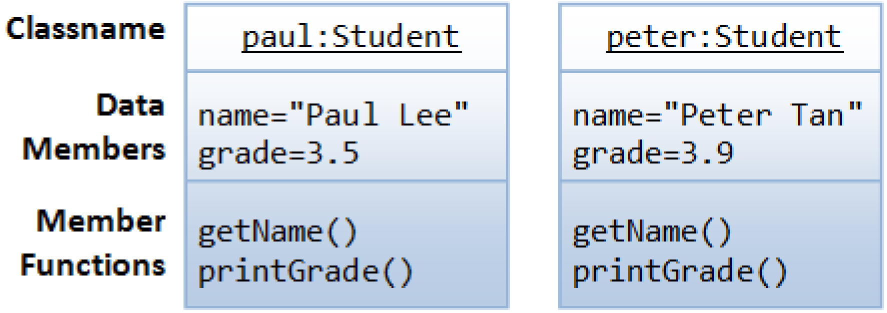
Classname (Identifier)	Student	Circle
Data Member (Static attributes)	name grade	radius color

Member Functions (Dynamic Operations)	getName()
	printGrade()
		getRadius()
		getArea()

SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed

run()
jump()
kickBall()
	move()
	park()
	accelerate()

Examples of classes



Classname	paul:Student	peter:Student

Data Members	name="Paul Lee"
	grade=3.5
		name="Peter Tan"
		grade=3.9

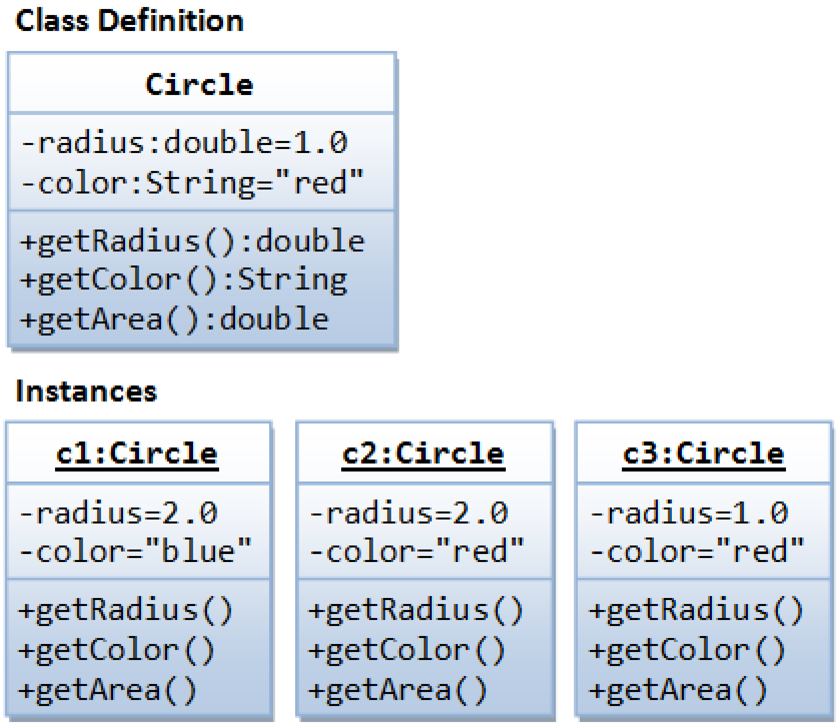
Member Functions	getName()
	printGrade()
		getName()
		printGrade()

Two instances of the `Student` class

Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language



Class Definition

Circle

-radius:double=1.0
-color:String="red"

+getRadius():double
+getColor():String
+getArea():double

Instances

c1:Circle

-radius=2.0
-color="blue"

+getRadius()
+getColor()
+getArea()

c2:Circle

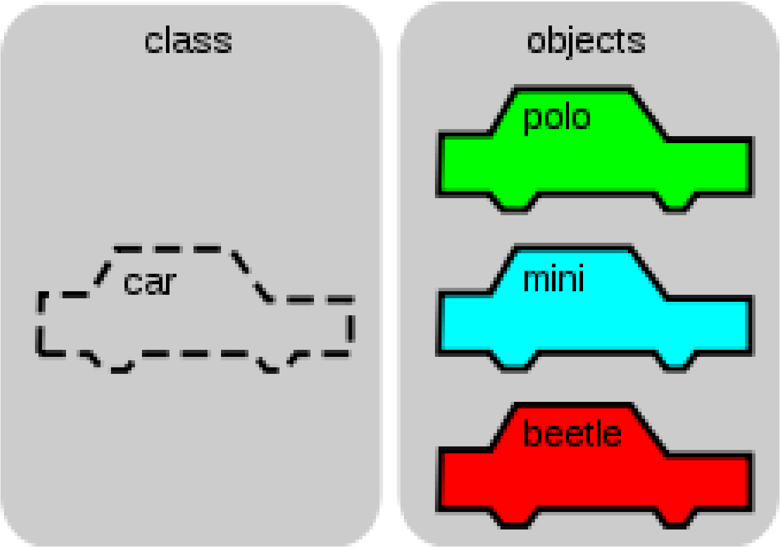
-radius=2.0
-color="red"

+getRadius()
+getColor()
+getArea()

c3:Circle

-radius=1.0
-color="red"

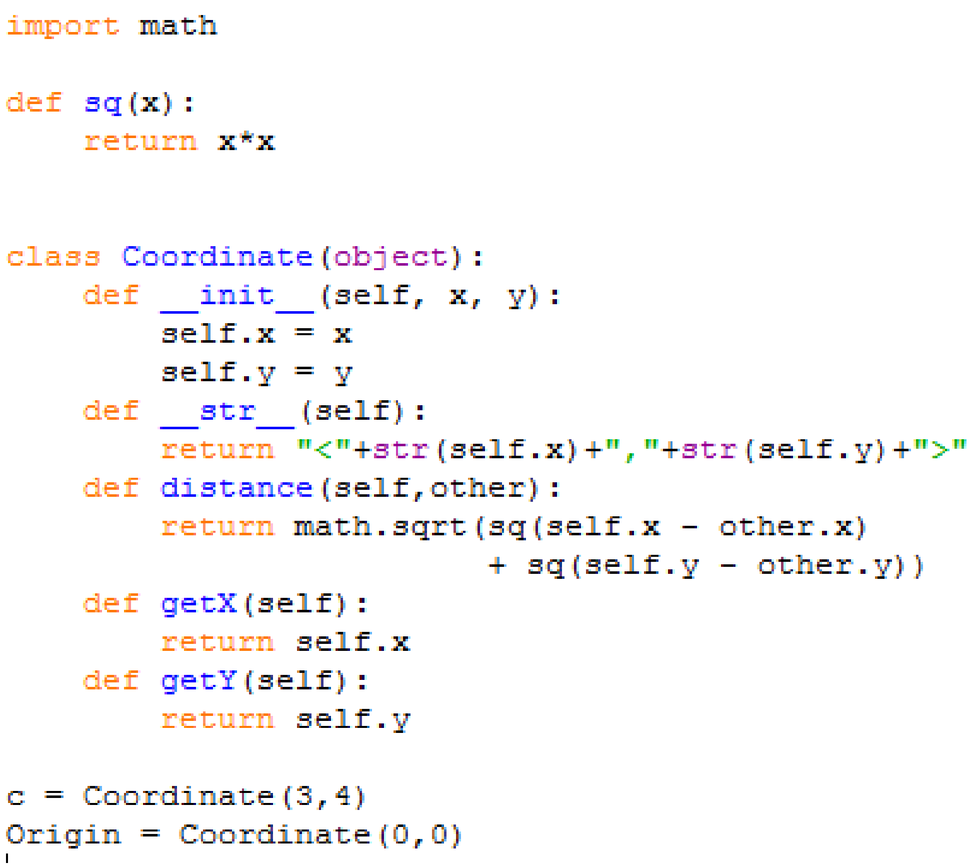
+getRadius()
+getColor()
+getArea()



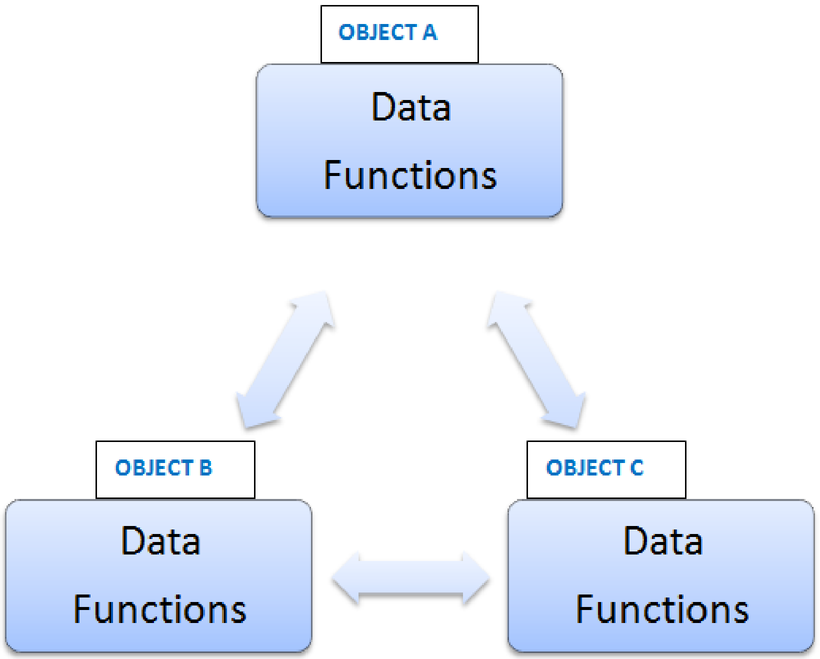
Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language



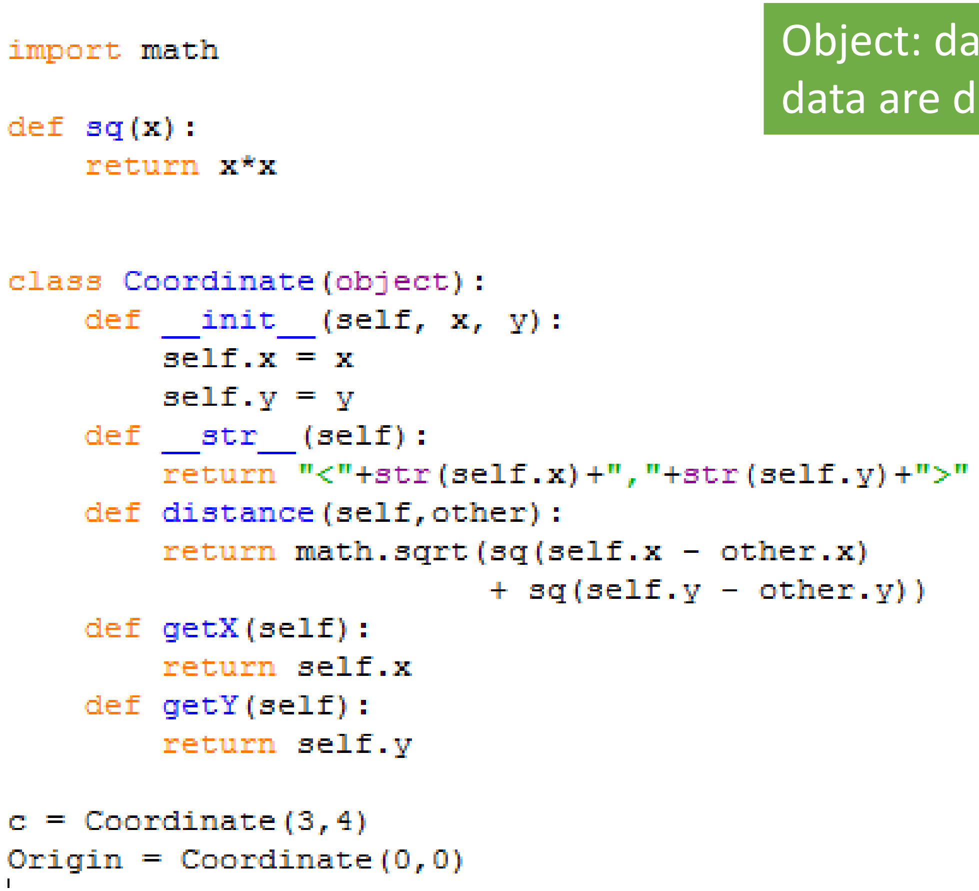
```
import math

def sq(x):
    return x*x


class Coordinate(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return "<"+str(self.x)+","+str(self.y)+">"
    def distance(self,other):
        return math.sqrt(sq(self.x - other.x)
                         + sq(self.y - other.y))
    def getX(self):
        return self.x
    def getY(self):
        return self.y

c = Coordinate(3,4)
Origin = Coordinate(0,0)
```



Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language



```
import math

def sq(x):
    return x*x


class Coordinate(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return "<"+str(self.x)+","+str(self.y)+">"
    def distance(self,other):
        return math.sqrt(sq(self.x - other.x)
                         + sq(self.y - other.y))
    def getX(self):
        return self.x
    def getY(self):
        return self.y

c = Coordinate(3,4)
Origin = Coordinate(0,0)
```

Object: data and methods of manipulating the data are designed and coded as a single unit

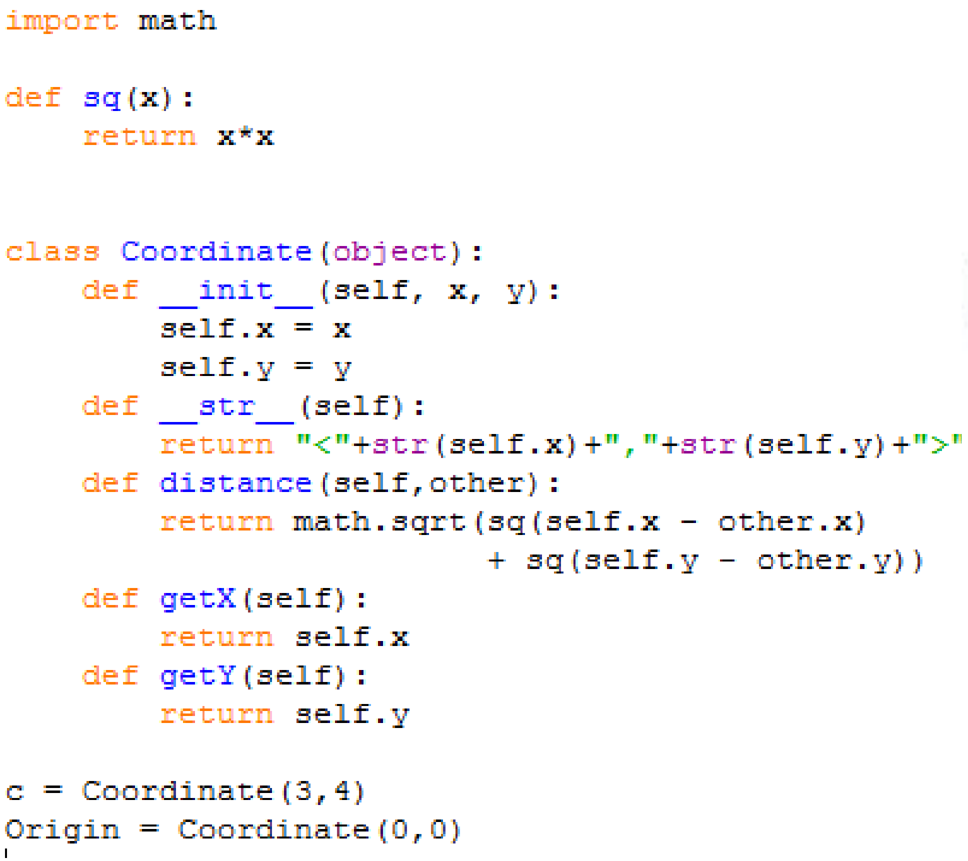
Object's Method: the only way that a user can access the data is via the Object's Method

- Code security
- Hiding details
- easy to modify

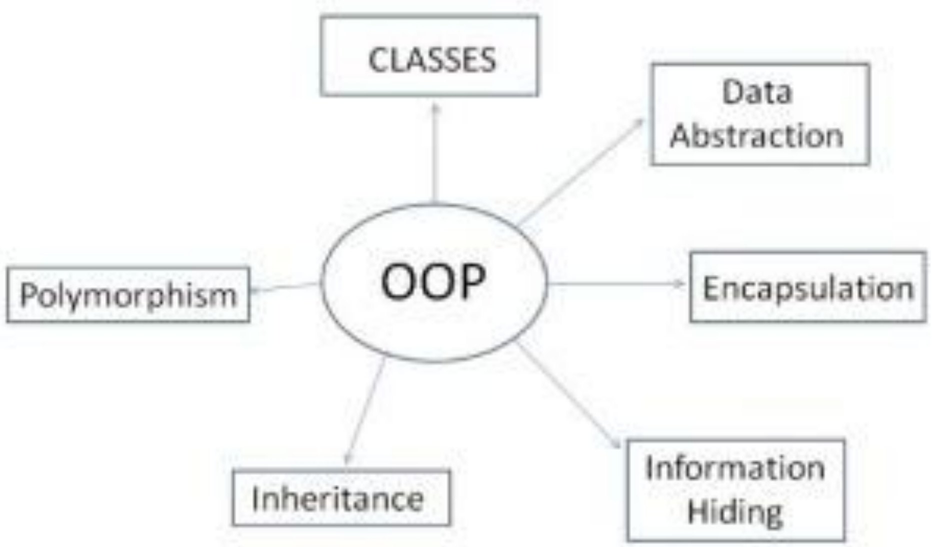
Characteristics of programming paradigms

Object-oriented paradigm

- Third-generation programming language
- Or High level programming language



```
import math

def sq(x):
    return x*x


class Coordinate(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return "<"+str(self.x)+","+str(self.y)+">"
    def distance(self,other):
        return math.sqrt(sq(self.x - other.x)
                         + sq(self.y - other.y))
    def getX(self):
        return self.x
    def getY(self):
        return self.y

c = Coordinate(3,4)
Origin = Coordinate(0,0)
```

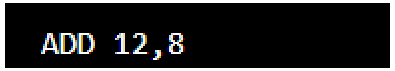


Characteristics of programming paradigms

Declarative paradigm

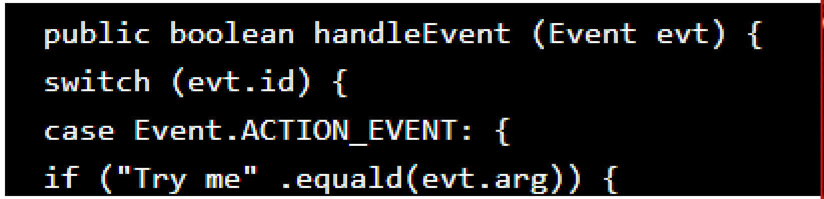
- Fifth-generation programming language
- Very/Advance High level programming language

2GL

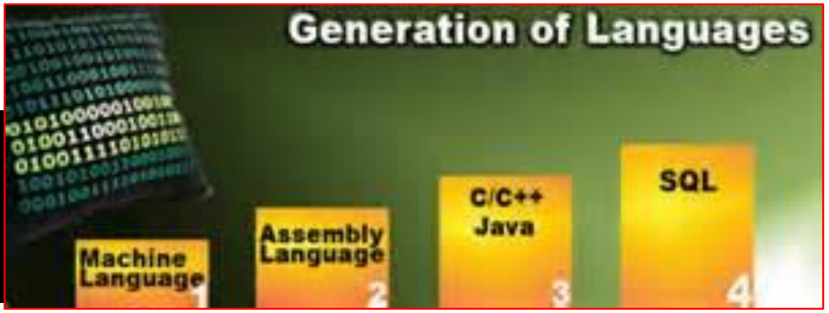


```
ADD 12,8
```

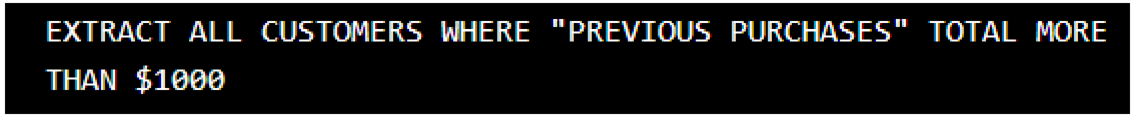
3GL



```
public boolean handleEvent (Event evt) {
switch (evt.id) {
case Event.ACTION_EVENT: {
if ("Try me" .equald(evt.arg)) {
```



4GL



```
EXTRACT ALL CUSTOMERS WHERE "PREVIOUS PURCHASES" TOTAL MORE
THAN $1000
```

5GL

https://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Fourth-generation_languages

designed to make the computer solve a given problem without the programmer. This way, the programmer only needs to worry about what problems need to be solved and what conditions need to be met, without worrying about how to implement a routine or algorithm to solve them. Fifth-generation languages are used mainly in artificial intelligence research. Prolog, OPS5, and Mercury are examples of fifth-generation languages

Characteristics of programming paradigms

Declarative paradigm

- Fifth-generation programming language
- Very/Advance High level programming language

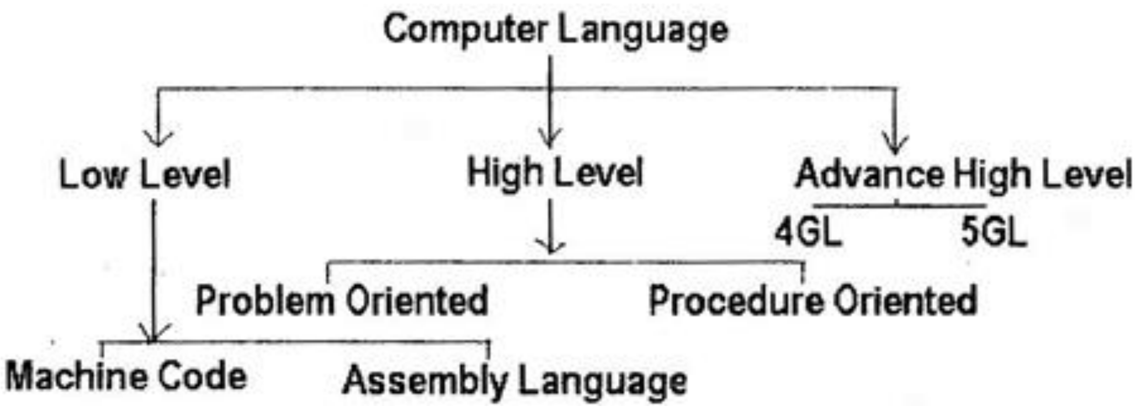
Procedure paradigm: encode a sequence of steps that determines ***how*** to solve the problem

Declarative paradigm: computer was told ***what*** the problem is, not how to solve it

given

Database or knowledge base of facts
A set of rules to apply the facts

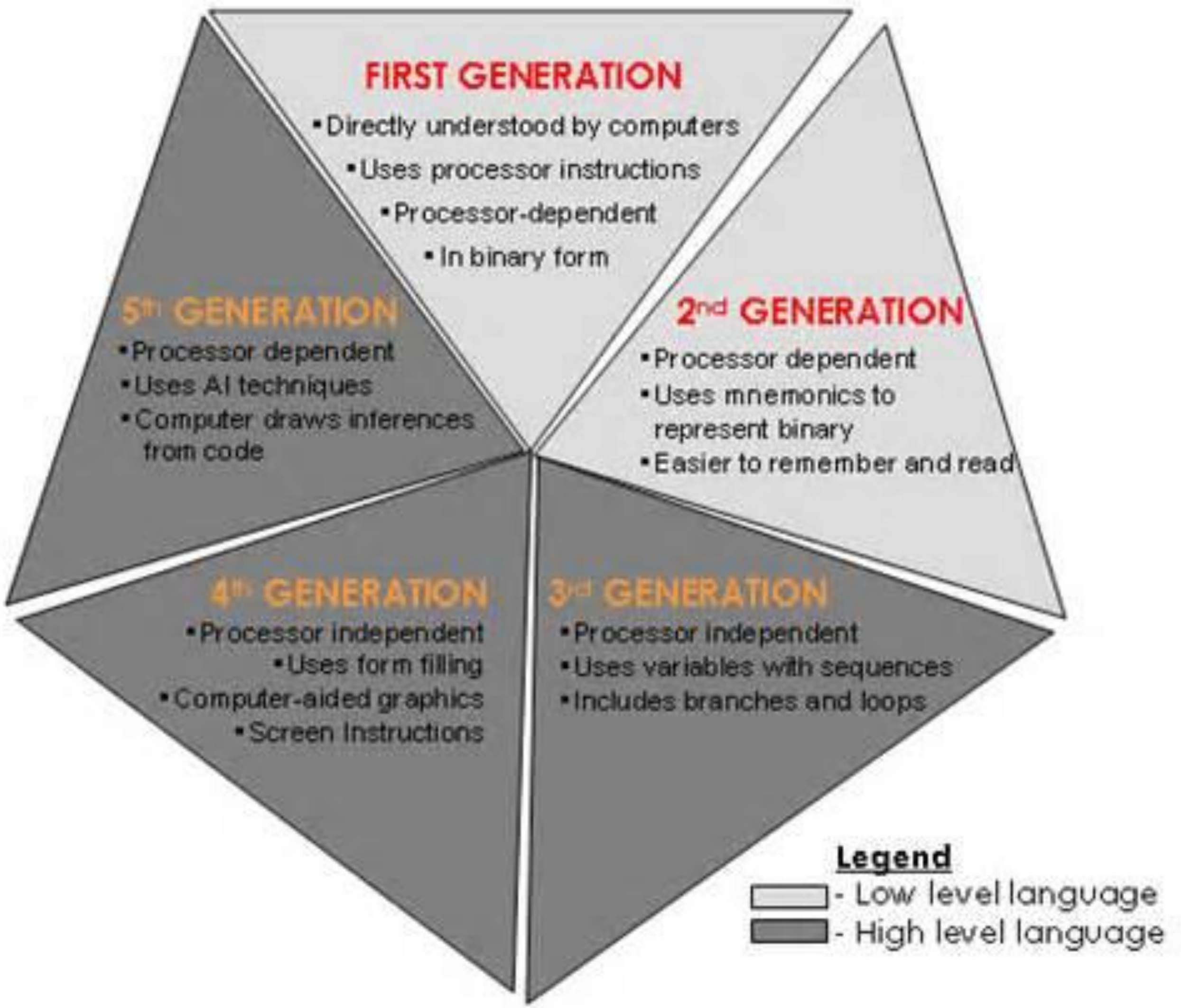
The computer program searches for a solution



OUTLINE

- Characteristics of programming paradigms
- Types of high-level language
- Structured program development
- Parameters and local and global variables
- Calling procedures and passing parameters via a stack

Types of high-level language



FIRST GENERATION

- Directly understood by computers
- Uses processor instructions
- Processor-dependent
- In binary form

2nd GENERATION

- Processor dependent
- Uses mnemonics to represent binary
- Easier to remember and read

3rd GENERATION

- Processor independent
- Uses variables with sequences
- Includes branches and loops

4th GENERATION

- Processor independent
- Uses form filling
- Computer-aided graphics
- Screen Instructions

5th GENERATION

- Processor dependent
- Uses AI techniques
- Computer draws inferences from code

Legend

□ - Low level language

■ - High level language

Types of high-level language

Procedural languages

- Specify how to solve a problem as a sequence of steps
- Use the constructs: sequence, selection and iteration

Activity

Find the area of a rectangle

Steps:

1. INPUT the length
2. INPUT the breadth
3. Multiply the length by the breadth and store the result
4. OUTPUT the result

C++

```
cout << "Enter the length: ";
cin >> Length;
cout << "Enter the breadth: ";
cin >> Breadth;
Area = Length * Breadth;
cout << "The area is " << Area << endl;
```

NOTE: logic matters, **task-oriented**
Programmer has to specify exactly what the program has to do

NOTE: may include functions and procedures but always specify the order in which instructions must be used to solve the problem

Types of high-level language

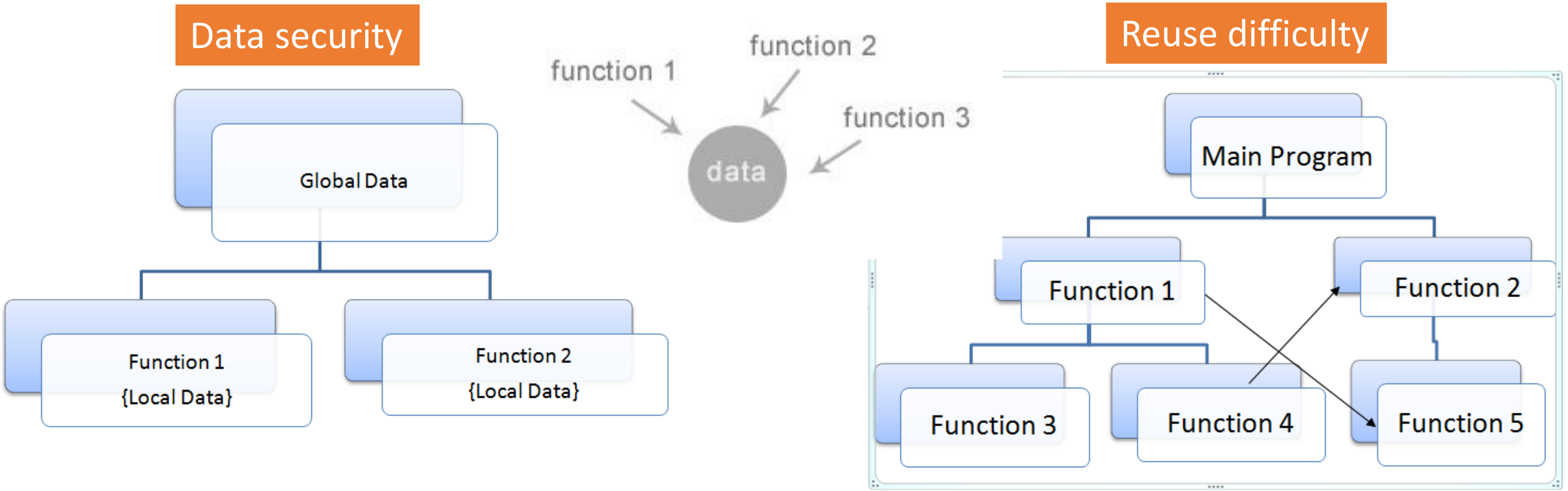
Procedural languages

- Specify how to solve a problem as a sequence of steps
- Use the constructs: sequence, selection and iteration

NOTE: function and procedures → help
there is danger of variables being altered inadvertently due to their scope being unclear

Data security

Reuse difficulty

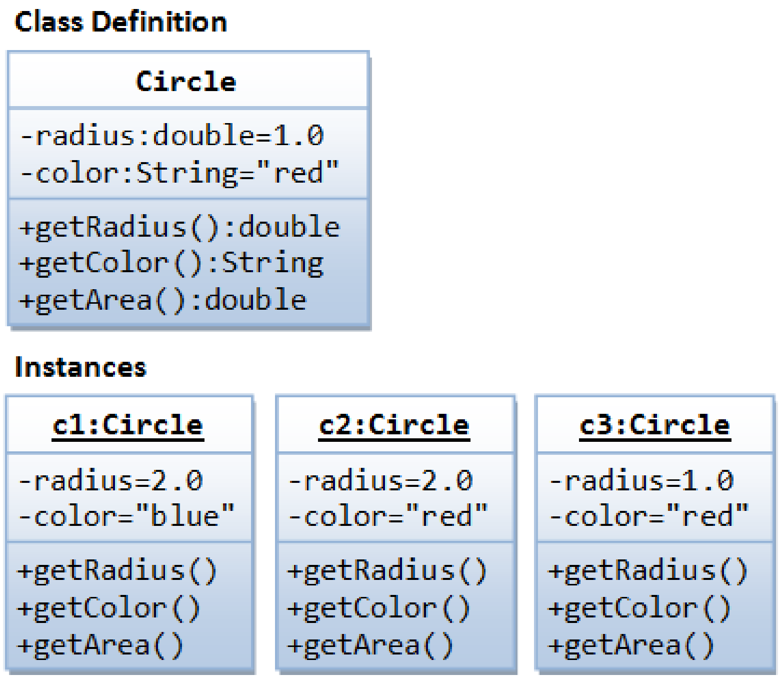


Types of high-level language

OOP

- Object-oriented programming languages

The real world consists of objects, not program



Class Definition

Circle

-radius:double=1.0
-color:String="red"

+getRadius():double
+getColor():String
+getArea():double

Instances

c1:Circle

-radius=2.0
-color="blue"

+getRadius()
+getColor()
+getArea()

c2:Circle

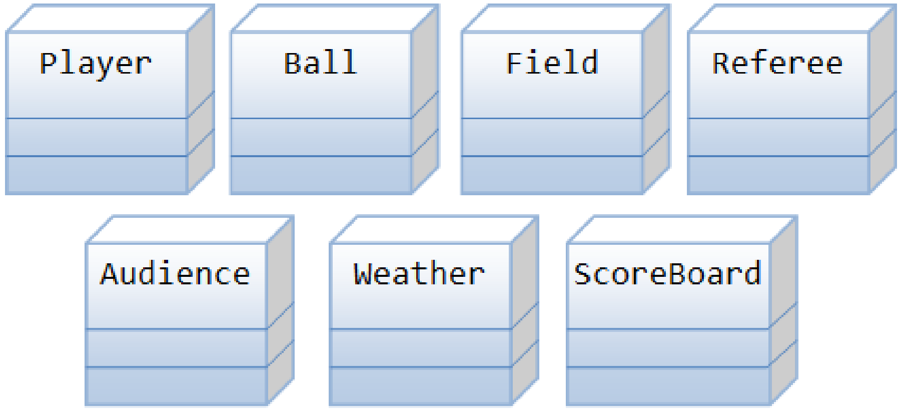
-radius=2.0
-color="red"

+getRadius()
+getColor()
+getArea()

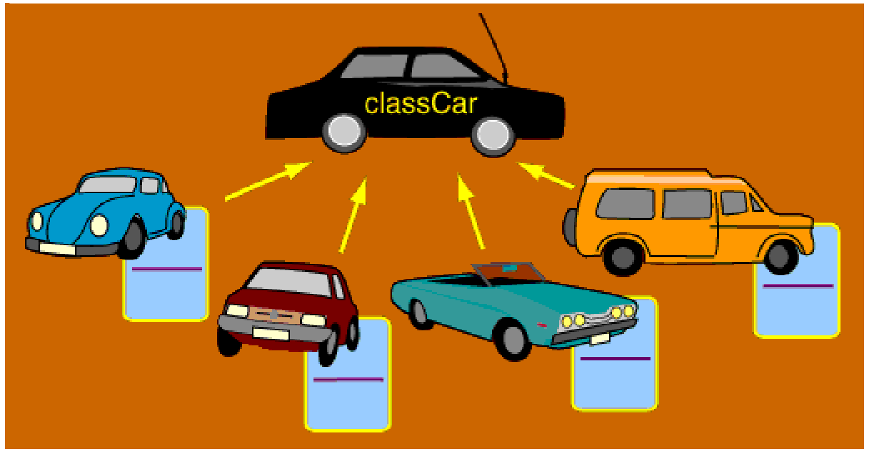
c3:Circle

-radius=1.0
-color="red"

+getRadius()
+getColor()
+getArea()



Classes (Entities) in a Computer Soccer Game

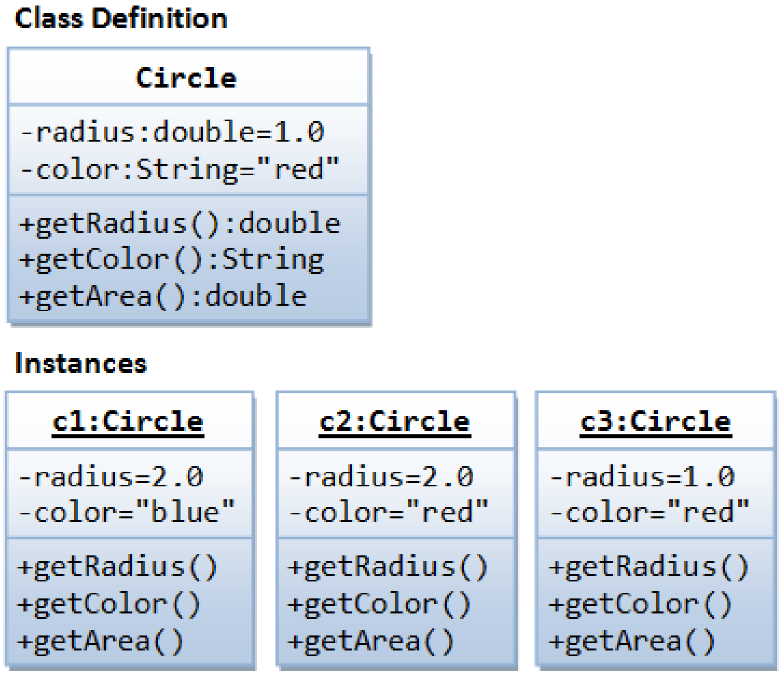


Types of high-level language

OOP

- Object-oriented programming languages

Class: blueprint or definition of some type of object
Object: an actual instance of the class
Can use data structures like array.



Class Definition

```
Circle
-radius:double=1.0
-color:String="red"
+getRadius():double
+getColor():String
+getArea():double
```

Instances

```
c1:Circle            c2:Circle           c3:Circle
-radius=2.0          -radius=2.0         -radius=1.0
-color="blue"        -color="red"        -color="red"
+getRadius()         +getRadius()        +getRadius()
+getColor()          +getColor()         +getColor()
+getArea()           +getArea()          +getArea()
```

Demo

Python supports many different kinds of data:

Types of high-level language

OOP

```
1234  int      3.14159  float   "Hello"   str
[1, 2, 3, 5, 7, 11, 13]        list
{"CA": "California", "MA": "Massachusetts"}
                              dict
```

Each of the above is an object.
Objects have:

- A type (a particular object is said to be an instance of a type)
- An internal data representation (primitive or composite)
- A set of procedures for interaction with the object

Example: [1,2,3,4]

- Type: list
- Internal data representation

– int length L, an object array of size S >= L, or

– A linked list of individual cells <data, pointer to next cell>

Reference: MIT 6.00X

Types of high-level language

OOP

Example: [1,2,3,4]

Class:

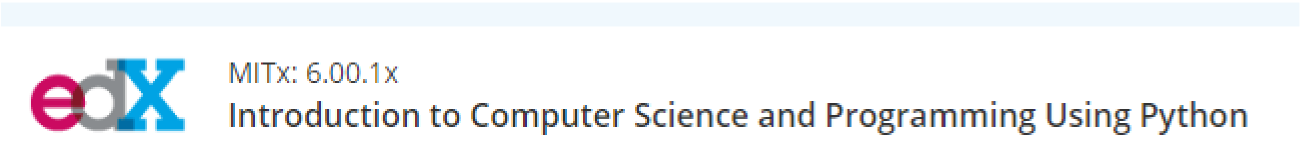
User-defined data type

- Type: list
- Internal data representation
  - int length L, an object array of size S >= L, or
  - A linked list of individual cells <data, pointer to next cell>
- Procedures for manipulating lists
  - l[i], l[i:j], l[i,j,k], +, *
  - len(), min(), max(), del l[i]
  - l.append(…), l.extend(…), l.count(…), l.index(…), l.insert(…), l.pop(…), l.remove(…), l.reverse(…), l.sort(…)

Reference: MIT 6.00X

Learning path

- https://levjj.github.io/thinkcspy/CMPS5P/l15.html
  - Point v.s. Turtle
- http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html (similar)
  - Point
- https://www.learnpython.org/en/Classes_and_Objects
  - MyClass



MITx: 6.00.1x

Introduction to Computer Science and Programming Using Python

Coordinate

Homework: ddl June 11 Thursday 8am

Programming practice:

Part A: https://levjj.github.io/thinkcspy/CMPS5P/l15.html

- complete 15.4,15.5, 15.6, 15.7, 15.8, 15.9 by running all the online code with codelens on(step running) to see what happened behind the code. Please save the screenshots of your running for each program in a word document. Submit the document.
- Study 15.10 for all key terms. Next lesson we will have a quiz on the concepts.
- Write python code for exercise 15.11. Please write the code all by yourself first! Submit the python code + the screenshots of your running results in your own IDE.

Part B: http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html

- Complete 15.12. Exercises question 5 and question 6.