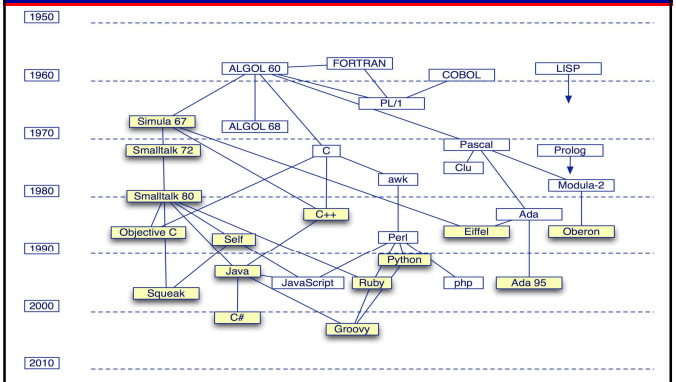
JAVA Programming
MCA 109
Pre-requisite based Study Material

Pre- Requisite Modules

- **Course on Java Programming, NPTEL**
  Speaker – Dr. Debasis Samanta, IIT Kharagpur
  Duration – 12 Weeks
  Link – https://nptel.ac.in/courses/106/105/106105191/
- **Lecture Slides**
  Pre-requisite based Study Material, Java Programming, by Dr. Ritika Wason

Evolution of Programming



Evolution of C

•C is a programming language developed in the 1970's alongside the UNIX operating system.

•C provides a comprehensive set of features for handling a wide variety of applications, such as systems development and scientific computation.

•C++ is an “extension” of the C language, in that most C programs are also C++ programs.

•C++, as opposed to C, supports “object-oriented programming.”

Evolution of C++

A “better C” that supports:

- •Systems programming
- •Object-oriented programming (classes & inheritance)
- •Programming-in-the-large (namespaces, exceptions)
- •Generic programming (templates)
- •Reuse (large class & template libraries

Need for C++

- Popular and relevant (used in nearly every application domain):
  - ▪end-user applications (Word, Excel, PowerPoint, Photoshop, Acrobat, Quicken, games)
  - ▪operating systems (Windows 9x, NT, XP; IBM’s K42; some Apple OS X)
  - ▪large-scale web servers/apps (Amazon, Google)
  - ▪central database control (Israel’s census bureau; Amadeus; Morgan-Stanley financial modeling)
  - ▪communications (Alcatel; Nokia; 800 telephone numbers; major transmission nodes in Germany and France)
  - ▪numerical computation / graphics (Maya)
  - ▪device drivers under real-time constraints
- Stable, compatible, scalable

C vs. C++

- C++ is C incremented
  (orig., "C with classes")
- C++ is more *expressive*
  (fewer C++ source lines needed than C source lines for same program)
- C++ is just as *permissive*
  (anything you can do in C can also be done in C++)
- C++ can be just as *efficient*
  (most C++ expressions need no run-time support;
  C++ allows you to
  - manipulate bits directly and interface with hardware without regard for safety or ease of comprehension, BUT
  - hide these details behind a safe, clean, elegant interface)
- C++ is more *maintainable*
  (*1000 lines of code* – even brute force, spaghetti code will work;
  100,000 lines of code – need good structure, or new errors will be introduced as quickly as old errors are removed)

Design Goals of C++

- Backward compatibility with C
  (almost completely – every program in K&R is a C++ program – but additional keywords can cause problems)
- Simplicity, elegance
  (few built-in data types, e.g., no matrices)
- Support for user-defined data types
  (act like built-in types; N.B. Standard Template Library (STL))
- No compromise in efficiency, run-time or memory
  (unless "advanced features" used)
- Compilation analysis to prevent accidental corruption of data
  (type-checking and data hiding)
- Support object-oriented style of programming

Java vs. C++

- No pointers — **just references**
- No functions — can declare `static` methods
- No global variables — use `public static` variables
- No destructors — **garbage collection** and `finalize`
- No linking — dynamic class loading
- No header files — can define `interface`
- No operator overloading — only method overloading
- No member initialization lists — call `super` constructor
- No preprocessor — `static final` **constants** and automatic in-lining

Java vs. C++

- no multiple inheritance — **implement multiple interfaces**
- no structs, unions, enums — **typically not needed**

Object Orientation Paradigm

- An approach to the solution of problems in which all **computations** are performed in context of objects.
- The objects are instances of **programming constructs**, normally called as **classes** which are **data abstractions** with **procedural abstractions** that operate on objects.
- A software system is a set of mechanism for performing certain **action** on certain data

  Algorithm + Data structure = Program
- **Data Abstraction + Procedural Abstraction**

Trade-offs of a Programming

- Ease-of-use versus power
- Safety versus efficiency
- Rigidity versus extensibility

Object & Classes

- **Class** is at core of Java
- Any concept implemented in Java prg is **encapsulated** within class
- Class define **new data type** which is used to create object of that type.

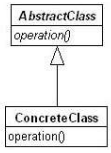
Classes – A Classification

A **Class** "is a set of objects that share a common structure and a common behavior." [Booch 1994].

Abstract Classes cannot be instantiated directly.

- The main purpose of an abstract class is to define a common interface for its subclasses.

Concrete Classes are not abstract and can have instances.



Defining Classes..

- Initializing data fields
  - By setting a value in a constructor
  - By assigning a value in the declaration
  - An initialization block
- When constructor is called
  - All data fields are initialized to their default values
  - All fields initializers and initialization blocks are executed, in the order in which they occur in the class declaration
  - If the first line of the constructor calls a second constructor, then the body of the second constructor is executed
  - The body of the constructor is executed

Object- *The CRUX of the matter!!*

- "An object is an **entity** which has a **state** and a defined set of **operations** which **operate** on that state."
- The **state** is represented as a set of object attributes. The operations associated with the object **provide services** to other objects (clients) which request these services when some computation is required
- Objects are **created** according to some **object class definition**. An object class definition serves as a **template** for objects. It includes **declarations** of all the attributes and services which should be associated with an object of that class.
- An Object is anything, **real** or **abstract**, about which we store data and those **methods** that manipulate the data.
- An **object** is a component of a program that knows how to perform certain actions and how to **interact** with other elements of the program.

Packages

- Grouping of classes
- Standard **Java packages** are inside java and javax
- A class can use all classes from its own package and all public classes from other packages
- **Import** a specific class or entire package using import statement
- Locating classes in package is an activity of package

Packages..

- **Static Imports**
  - In Java SE 5.0, import statement enhanced to import static methods & fields

    ```
    import static java.lang.System.*;
    out.println("---");
    ```
  - Two practical uses
    - ✓Mathematical functions: static import of Math class

      ```
      sqrt(pow(x,2)+pow(y,2))
      Math.sqrt(Math.pow(x,2)+Math.pow(y,2))
      ```
    - ✓Cumbersome constants

      ```
      if (d.get(DAY_OF_WEEK) == MONDAY)
      if (d.get(Calender.DAY_OF_WEEK) == Calender.MONDAY)
      ```

Packages..

- Import ONLY imports public classes from the specified package
- Classes which are not public cannot be referenced from outside their package.
- There is no way to "import all classes except one"
  - import either imports a single class or all classes within the package
  - Note: importing has no runtime or performance implications.
  - It is only importing a namespace so that the compiler can resolve class names.

Object Orientation Criteria

Object orientation adapts to the following criteria's-

1. Changing requirements
2. Easier to maintain
3. More robust
4. Promote greater design
5. Code reuse
6. Higher level of abstraction
7. Encouragement of good programming techniques
8. Promotion of reusability

Object Orientated Features

1. **OBJECT -** Object is a collection of number of **entities**. Objects take up space in the memory. Objects are **instances of classes**. When a program is executed , the objects interact by sending messages to one another. Each object contain **data** and **code** to manipulate the data. Objects can interact without having know details of each others data or code. **Each instance** of an object can hold its own **relevant data**.

2. **CLASS -** Class is a **collection** of **objects** of similar type. Objects are **variables** of the **type class**. Once a class has been defined, we can create any number of objects belonging to that class. Classes are **user define data types**. A class is a **blueprint** for any functional entity which defines its **properties** and its **functions**.

Object Orientated Features

3. DATA ENCAPSULATION – Combining data and functions into a single unit called **class** and the process is known as **Encapsulation**. Class variables are used for storing data and functions to specify various operations that can be performed on data. This process of **wrapping up** of data and functions that operate on data as a single unit is called as data encapsulation. Data is **not accessible** from the outside world and only those function which are present in the class can access the data.

4. DATA ABSTRACTION- Abstraction (from the Latinn *abs* means *away from* and *trahere* means to draw) is the **process** of taking away or **removing characteristics** from something in order to reduce it to a **set of essential characteristics.** Advantage of data abstraction is **security**.

Object Orientated Features

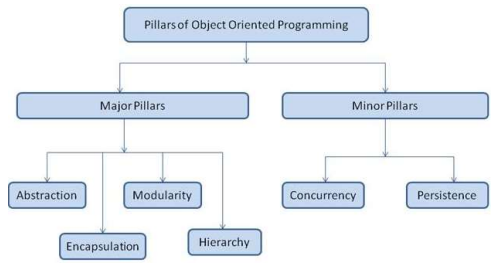
5. INHERITANCE- It is the process by which object of one class **acquire** the **properties** or features of objects of another class. The concept of inheritance provide the idea of reusability means we can add **additional features** to an existing class **without modifying it.** This is possible by driving a new class from the existing one. **Advantage** of inheritance is **reusability** of the **code**.

6. MESSAGE PASSING - The process by which **one object** can interact with **other object** is called **message passing**.

7. **POLYMORPHISM -** A greek term means **ability to take more than one form.** An operation may exhibit different behaviours in different instances. The behaviour depends upon the **types of data** used in the operation.

Object Orientated Features

8. PERSISTENCE - The process that allows the state of an **object** to be saved to **non-volatile storage** such as a file or a database and later restored even though the original creator of the object no longer exists.



Inheritance..

- It is the responsibility of the subclass constructor to invoke the appropriate superclass constructors
- Superclass constructors can be called using the "**super**" keyword in a manner similar to "**this**"
- It must be the first line of code in the constructor
- If a call to super is not made, the system will automatically attempt to invoke the no-argument constructor of the superclass.
- **Super has two general forms.**
  - The first calls the superclass constructor
  - The second is used to access a member of the superclass that has been hidden by a member of a subclass
- A superclass reference can refer to an instance of the superclass OR an instance of ANY class which inherits from the superclass.
- Dynamic Method Dispatch will be applicable