Unit II
Structuring the Data, Computations and Program

Elementary Data Types : Primitive data Types, Character String types, User Defined Ordinal Types, Array types, Associative Arrays, Record Types, Union Types, Pointer and reference Type.
Expression and Assignment Statements: Arithmetic expression, Overloaded Operators, Type conversions, Relational and Boolean Expressions, Short Circuit Evaluation, Assignment Statements, Mixed mode Assignment.
Statement level Control Statements: Selection Statements, Iterative Statements, Unconditional Branching.
Subprograms: Fundamentals of Sub Programs, Design Issues for Subprograms, Local referencing Environments, Parameter passing methods.
Abstract Data Types and Encapsulation Construct: Design issues for Abstraction, Parameterized Abstract Data types, Encapsulation Constructs, Naming Encapsulations.

Abstract Data Types and Encapsulation Construct

1. Introduction to Abstract Data Types

Abstraction is one of the key concepts of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

That’s a very generic concept that’s not limited to object-oriented programming. You can find it everywhere in the real world.

Abstraction in the real world

I’m a coffee addict. So, when I wake up in the morning, I go into my kitchen, switch on the coffee machine and make coffee. Sounds familiar?

Making coffee with a coffee machine is a good example of abstraction.

You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get.

The thing you don’t need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee. You don’t need to know the ideal temperature of the water or the amount of ground coffee you need to use.

Someone else worried about that and created a coffee machine that now acts as an abstraction and hides all these details. You just interact with a simple interface that doesn’t require any knowledge about the internal implementation.

You can use the same concept in object-oriented programming languages like Java.

Abstraction in OOP

Objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation. But you don't need to understand how this method is implemented and which kinds of actions it has to perform to create the expected result.

Different Types of Abstraction

There are primarily two types of abstraction implemented in OOPs. One is data abstraction which pertains to abstracting data entities. The second one is process abstraction which hides the underlying implementation of a process. Let's take a quick peek into both of these.

Data Abstraction

Data abstraction is the simplest form of abstraction. When working with OOPS, you primarily work on manipulating and dealing with complex objects. This object represents some data but the underlying characteristics or structure of that data is actually hidden from you. Let's go back to our example of making coffee.

Let's say that I need a special hazelnut coffee this time. Luckily, there's a new type of coffee powder or processed coffee beans that already have hazelnut in it. So I can directly add the hazelnut coffee beans and the coffee machine treats it as just any other regular coffee bean. In this case, the hazelnut coffee bean itself is an abstraction of the original data, the raw coffee beans. I can use the hazelnut coffee beans directly without worrying about how the original coffee beans were made to add the hazelnut flavor to it.

Therefore, data abstraction refers to hiding the original data entity via a data structure that can internally work through the hidden data entities. As programmers, we don't need to know what the underlying entity is, how it looks etc.

Process Abstraction

Where data abstraction works with data, process abstraction does the same job but with processes. In process abstraction, the underlying implementation details of a process are hidden. We work with abstracted processes that under the hood use hidden processes to execute an action.

Circling back to our coffee example, let's say our coffee machine has a function to internally clean the entire empty machine for us. This is a process that we may want to do every once a week or two so that our coffee machine stays clean. We press a button on the machine which sends it a command to internally clean it. Under the hood, there is a lot that will happen now. The coffee machine will need to clean the piston, the outlets or nozzles from which it pours the coffee, and the container for the beans, and then finally rinse out the water and dry out the system.

A single process of cleaning the coffee machine was known to us, but internally it implements multiple other processes that were actually abstracted from us. This is process abstraction in a nutshell.

Data Abstraction for Built-in Data Types

(e.g. floating-point type)

User-Defined Abstract Data Types

(e.g. structure, classes)

Benefits of Abstract Data Types:

i. Increased Reliability

ii. It reduces the range of code and number of variables of which a programmer must be aware when writing or reading a part of the program.

iii. Makes name conflicts less likely, because the scope of variables is smaller

2. Design issues for Abstraction

1. The form of the container for the interface to the type.

2. Whether abstract data types can be parameterized?

3. What access controls are provided and how such controls are specified?

4. Whether the specification of the type is physically separate from its implementation (or whether that is a developer choice).

Unit II
Structuring the Data, Computations and Program

3. Parameterized Abstract Data types

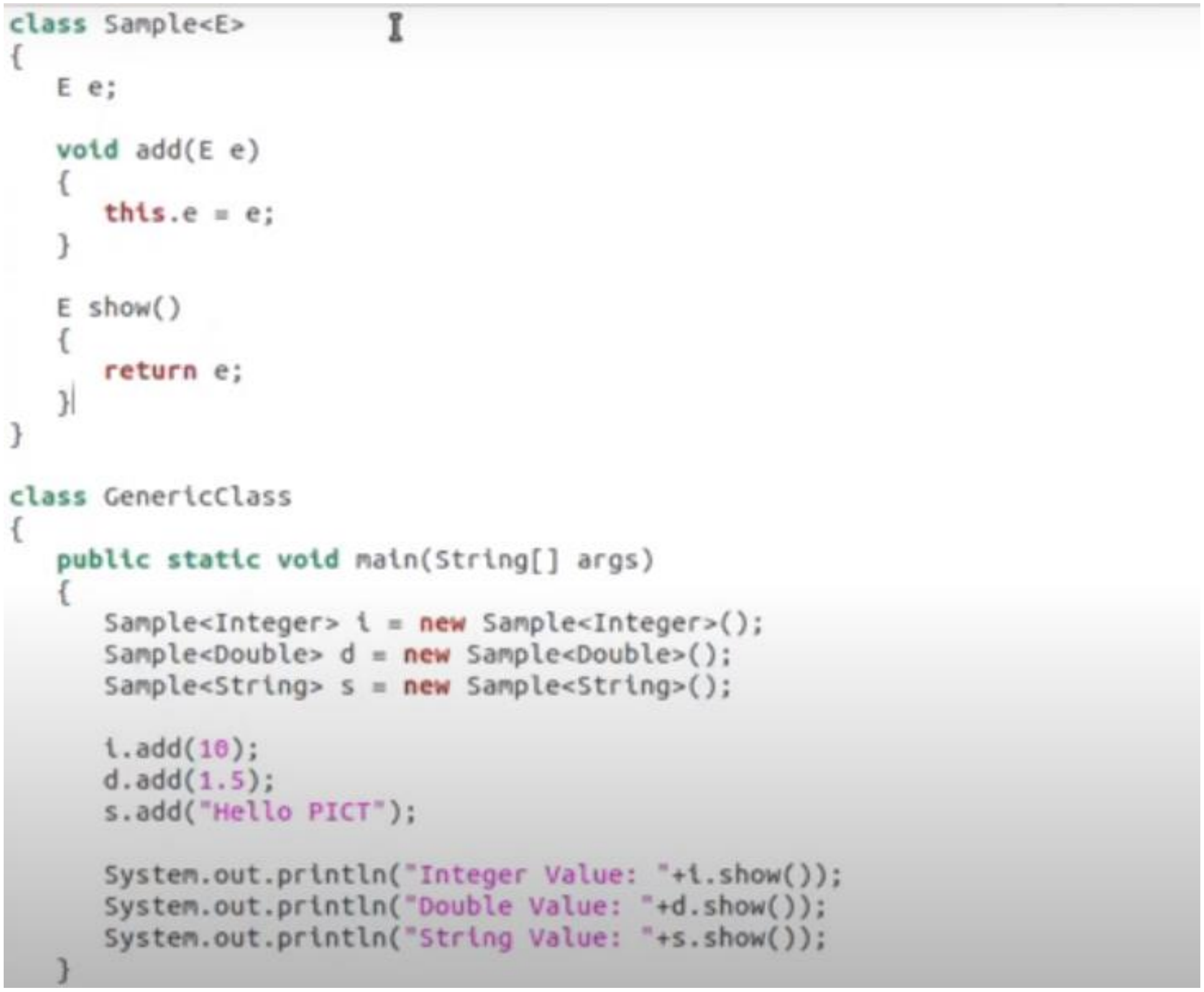
To pass the parameters to the Abstract Data Types.

To generalize the types of parameters passed to the Abstract Data Types, following features are provided:

i. Templates in C++

```
// C++ program to implement function templates
#include <iostream>
using namespace std;

// Function template to swap two numbers
template <class T>
int swap_numbers(T& x, T& y)
{
   T t;
   t = x;
   x = y;
   y = t;
   return 0;
}

int main()
{
   int a, b;
   a = 10, b = 20;

   float c,d;
   c = 10.50, d = 20.50;

   // Invoking the swap()
   swap_numbers(a, b);
   cout << a << " " << b << endl;

// Invoking the swap()
   swap_numbers(c, d);
   cout << c << " " << d << endl;

   return 0;
}
```

ii. Generic Classes in Java



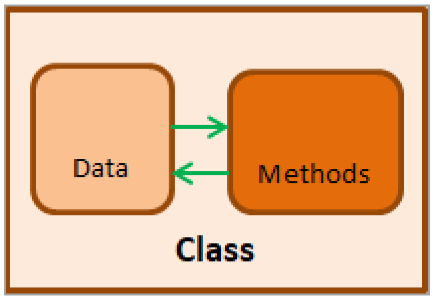
```
class Sample<E>
{
   E e;

   void add(E e)
   {
      this.e = e;
   }

   E show()
   {
      return e;
   }
}

class GenericClass
{
   public static void main(String[] args)
   {
      Sample<Integer> i = new Sample<Integer>();
      Sample<Double> d = new Sample<Double>();
      Sample<String> s = new Sample<String>();

      i.add(10);
      d.add(1.5);
      s.add("Hello PICT");

      System.out.println("Integer Value: "+i.show());
      System.out.println("Double Value: "+d.show());
      System.out.println("String Value: "+s.show());
   }
```

What are the differences between encapsulation and abstraction?

Abstraction is a method of removing the unnecessary details in the code and focusing on the relevant parts. For example, instead of writing a function that multiplies two numbers, we can write a function “multiply” and use it to multiply any number with any number. Abstraction is a way of representing some specific data. Encapsulation is a way of hiding the complexity of something and exposing only the parts you wish to expose. For example, if you have a class that has one or more private fields that you use to store the data, then you are in encapsulation. The class contains the data and has methods that expose the public data.

4. Encapsulation Constructs

Encapsulation is a kind of “data hiding”



The above figure represents a class which is an encapsulation unit that bundles the data and methods operating on this data into a single unit.

We can visualize encapsulation as a medical capsule. As we all know the medicine is enclosed inside a medical capsule. Similarly, data and methods are enclosed in a single unit in encapsulation.

Thus encapsulation acts as a protective shield around the data and prevents the data from unauthorized access by the outside world. In other words, it protects the sensitive data of our application.

i. Encapsulation in C language using Header Files

ii. Encapsulation in C++

a. Header Files

b. Classes

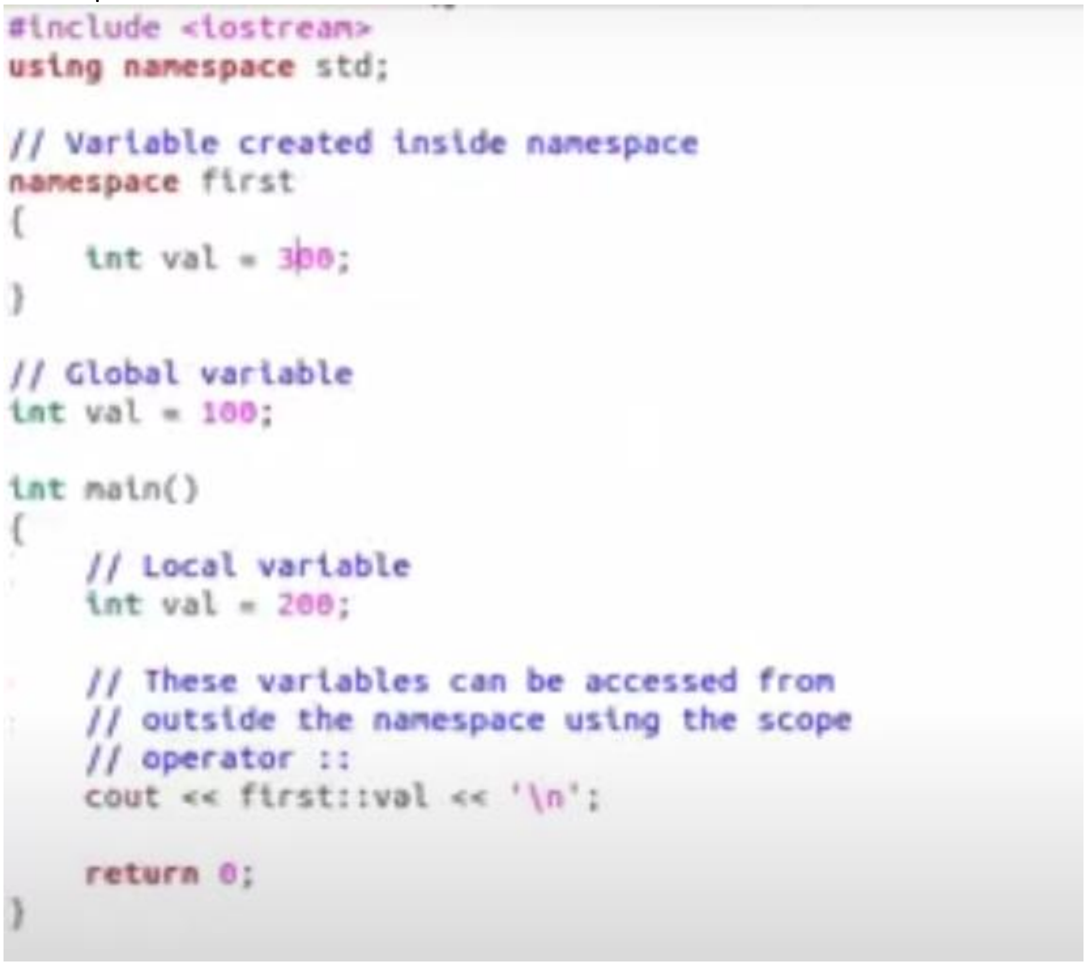
iii. In Java, there are two steps to implement encapsulation. Following are the steps:

- Use the access modifier ‘private’ to declare the class member variables.
- To access these private member variables and change their values, we have to provide the public getter and setter methods respectively.

5. Naming Encapsulations

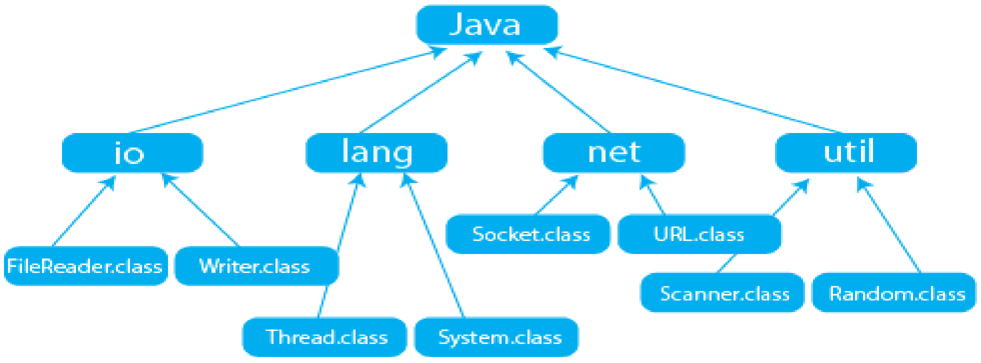
How to avoid naming conflicts...

i. Namespaces in C++



```
#include <iostream>
using namespace std;

// Variable created inside namespace
namespace first
{
    int val = 300;
}

// Global variable
int val = 100;

int main()
{
    // Local variable
    int val = 200;

    // These variables can be accessed from
    // outside the namespace using the scope
    // operator ::
    cout << first::val << '\n';

    return 0;
}
```

ii. Packages in Java
Create packages with user defined data and classes. User can later import the packages to use them.
Let’s see how the hierarchy of the java build-in package looks like



```
import java.util.Scanner;
class PrepBytes {
   public static void main(String[] args) {
      Scanner sc=new Scanner(System.in);
      int num=sc.nextInt();
      System.out.println("The Number is "+num);
   }

}
```

```
package mypackage;
class PrepBytes {
   public static void main(String[] args) {
      System.out.println("This is user-defind package");
   }
}
```