**Chapter # 03: Getting Deeper with Class Object**

**3.1 Objects of Class:**

In Java, objects are instances of a class, which encapsulate both the data (attributes) and behavior (methods) of the class. Objects are created using the new keyword, which allocates memory for the object and initializes its attributes to their default values.

Here's an example of a simple Person class and how to create an object of that class:

|  |
| --- |
| Sample Objects Of Class |
| public class Person {      private String name;      private int age;      public Person(String name, int age) {          this.name = name;          this.age = age;      }      public void sayHello() {          System.out.println("Hello, my name is " + name);      }  }  // Creating an object of the Person class  Person p = new Person("Alice", 30);  p.sayHello(); // prints "Hello, my name is Alice" |

Java objects make it possible to package data and behavior into reusable and adaptable pieces, which can make designing and implementing complicated systems easier. You can develop custom types that meet the requirements of your application and represent real-world entities naturally and logically by defining classes and producing objects of those classes.

**3.2 Static and Dynamic Functions/Variables:**

Static and dynamic functions and variables are concepts in computer programming that refer to how data and functions are defined and used in a program.

* **Static Variables/Function:**

Static functions and variables are defined as part of a class or object, and they exist independently of any instance of that class or object. They are stored in a fixed location in memory and retain their values throughout the program's lifetime.

Static functions and variables are typically used for values that do not change during program execution, such as program-wide constants or utility functions that do not depend on the state of the object. Static functions and variables in Java are accessed by using the class name rather than creating an object of the class.

Static variables and functions are associated with the class itself rather than with any particular instance of the class. This means that all objects of the class share the same static variable or function. They are initialized when the class is loaded and are available throughout the lifetime of the program.

Static variables and functions can be accessed using the class name, without the need to create an object of the class. For example, in the following code, the static variable counter and the static function incrementCounter() are associated with the MyClass class:

|  |
| --- |
| public class MyClass {      static int counter = 0;      static void incrementCounter() {          counter++;      }  } |

To access the static variable or function, you can use the following syntax:

|  |
| --- |
| MyClass.counter = 10;  MyClass.incrementCounter(); |

* **Dynamic Functions/Variables:**

Dynamic variables and functions are those that come with an instance of a class or object. They are destroyed when the instance of the class is destroyed and only exist when an instance of the class is created. Usually, dynamic functions and variables are used to hold values that are unique to a given instance of the class, such as an object's state or data that is produced while a program is being executed.

An object of the class is used to access dynamic functions and variables. Java uses the word "this" to denote the current instance of the class, and the dot notation is used to access dynamic functions and variables.

Dynamic variables and functions are associated with the instance of the class. Each instance of the class has its own copy of the dynamic variable and can call the dynamic function independently of other instances of the class.

Dynamic variables and functions can be accessed using an object of the class. For example, in the following code, the dynamic variable x and the dynamic function printX() are associated with each instance of the MyClass class:

|  |
| --- |
| public class MyClass {      int x;      void printX() {          System.out.println(x);      }  } |

To access the dynamic variable or function, you need to create an object of the class, as follows:

|  |
| --- |
| MyClass obj1 = new MyClass();  obj1.x = 5;  obj1.printX(); // Ouputs 5  MyClass obj2 = new MyClass();  obj2.x = 10;  obj2.printX(); // Outputs 10 |

**3.3 Class Design:**

When it comes to designing classes, there are a few key principles that can help guide the process:

* **Identify the responsibilities and behaviors of the class:**

You must be completely aware of the purpose of the class before you can begin to create it. What duties does it have? What actions should it take? What issues should it address? This might assist you in determining the course's objectives and ensuring that it has a distinct purpose.

* **Consider the relationships between classes:**

It's crucial to take into account how different classes in the system interact with one another because classes don't exist in a vacuum. How is the class dependent on other entities? With which other classes does it work together? You can create a class that is logically and effectively integrated into the wider system by understanding these relationships.

* **Use abstraction to simplify complexity:**

Classes can easily grow complex, thus it's crucial to employ abstraction to make their design simpler. You may reduce duplication and improve modularity by defining a set of shared behaviors that can be shared by different classes with the aid of abstract classes and interfaces.

* **Ensure cohesion and encapsulation:**

A well-designed class will have cohesiveness, which is the quality of having all of its duties tied to a single, clearly stated goal. Encapsulation is crucial since it enables you to keep class internals private and only disclose what's required for use by other classes.

* **Test early and often:**

Testing is an important part of class design, as it allows you to identify and correct issues early in the development process. As you design your class, think about how you'll test it and what scenarios you'll need to cover to ensure that it works as expected.

Overall, class design is a complex process that requires careful consideration of many factors. By following these principles, you can create classes that are well-structured, maintainable, and easy to use.