Lecture 8

Polymorphism

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

- **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Polymorphism

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

- There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

- If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism

- **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

- Upcasting: If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

Runtime Polymorphism

- A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

- We are accessing the data member by the reference variable of Parent class which refers to the subclass object. Since we are accessing the data member which is not overridden, hence it will access the data member of the Parent class always.

Static Binding and Dynamic Binding

- Connecting a method call to the method body is known as binding. There are two types of binding
  - Static Binding (also known as Early Binding).
  - Dynamic Binding (also known as Late Binding).

Static Binding and Dynamic Binding

- Static binding: When type of the object is determined at compiled time(by the compiler), it is known as static binding. If there is any private, final or static method in a class, there is static binding.

- Dynamic binding: When type of the object is determined at run-time, it is known as dynamic binding.

instanceof

- The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

- The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Inner Classes

- In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

- To access the inner class, create an object of the outer class, and then create an object of the inner class

Private & Static inner class

- **Private Inner Class:** Unlike a "regular" class, an inner class can be **private** or **protected**. If you don't want outside objects to access the inner class, declare the class as **private**

- **Static Inner Class:** An inner class can also be **static**, which means that you can access it without creating an object of the outer class.

- One advantage of inner classes, is that they can access attributes and methods of the outer class