typescript Copy code

```typescript
class Animal {
    public void makeSound() {
        System.out.println("Animal is making a sound");
    }
}

class Dog extends Animal {
    // Method overriding
    @Override
    public void makeSound() {
        System.out.println("Bark!");
    }

    // Method overloading
    public void makeSound(String breed) {
        System.out.println(breed + " is barking");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.makeSound(); // Output: Animal is making a sound

        Dog dog = new Dog();
        dog.makeSound(); // Output: Bark!

        dog.makeSound("Labrador"); // Output: Labrador is barking
    }
}
```

Abstract Class vs Interface:

- An abstract class can have both abstract and non-abstract methods, while an interface can only have abstract methods.
- An abstract class can have instance variables and constructors, while an interface cannot.
- A class can extend only one abstract class, while it can implement multiple interfaces.
- An abstract class provides default implementation for its methods, while an interface only declares the method signatures and does not provide any implementation.

Here's an example of implementing an interface using an anonymous inner class:

csharp Copy code

```csharp
interface Greeting {
   void greet();
}

public class Example {
   public static void main(String[] args) {
      Greeting greeting = new Greeting() {
         public void greet() {
            System.out.println("Hello from anonymous inner class!");
         }
      };
      greeting.greet();
   }
}
```

In this example, we define a simple interface `Greeting` with a single method `greet()`. Then in the `main()` method of the `Example` class, we create an instance of the `Greeting` interface using an anonymous inner class. This allows us to define the implementation of the `greet()` method inline, without having to create a separate class. We then call the `greet()` method on this instance to print the greeting message.

Packages in Java:

- Packages are used to organize related classes and interfaces into a single unit.
- Packages help in avoiding naming conflicts in large projects and make it easier to manage code.
- Java provides two types of packages: built-in packages and user-defined packages.
- Built-in packages are already available in Java, such as java.lang, java.util, etc.
- User-defined packages are created by the developer as per the requirement of the project.

Static Import in Java:

- The static import statement is used to import static members of a class directly into the current class.
- It simplifies the code by allowing you to use static members without specifying the class name.
- To use the static import statement, the members must be defined as public static in the class.
- It helps to reduce the verbosity of the code and make it more readable.

Example program for static import:

arduino Copy code

```arduino
import static java.lang.Math.*;

class Main {
  public static void main(String[] args) {
    double x = 4.0;
    double y = sqrt(x);
    double z = pow(x, 2);
    System.out.println("Square root of " + x + " is " + y);
    System.out.println(x + " raised to the power of 2 is " + z);
  }
}
```

Object-Oriented Programming (OOP) is a programming paradigm that focuses on organizing software design around objects that contain data and behavior. The features of OOP are:

1. Encapsulation: Encapsulation is a mechanism that binds data and code together into a single unit called a class. It helps in hiding the implementation details from the outside world and protects the data from unauthorized access.
2. Abstraction: Abstraction is the process of hiding the implementation details and showing only the essential features of the object. It helps in reducing the complexity of the system and makes it easy to understand and use.
3. Inheritance: Inheritance is a mechanism that allows a class to inherit the properties and behavior of another class. It helps in reusing the code and creating a hierarchy of classes.
4. Polymorphism: Polymorphism is the ability of an object to take on many forms. It allows objects to be treated as instances of their own class or as instances of their parent class.
5. Classes and Objects: Classes and Objects are the basic building blocks of Object-Oriented Programming. A class is a blueprint for creating objects, while an object is an instance of a class.
6. Overloading and Overriding: Overloading is the process of defining multiple methods with the same name in the same class, but with different parameters. Overriding is the process of defining a method in a subclass that is already defined in the parent class.
7. Interfaces: Interfaces are a way of achieving abstraction in Java. An interface is a collection of abstract methods that can be implemented by any class that implements the interface.

These features of OOP help in creating modular, reusable, and maintainable code.

```java
// Base class or superclass
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }

    void sleep() {
        System.out.println(name + " is sleeping.");
    }
}

// Derived class or subclass
class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age); // calling superclass constructor
        this.breed = breed;
    }

    void bark() {
        System.out.println(name + " is barking.");
    }
}

// Main class
class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");
        myDog.eat(); // calling inherited method from Animal class
        myDog.sleep(); // calling inherited method from Animal class
        myDog.bark(); // calling method from Dog class
    }
}
```