1. **Code reusability**: Inheritance allows a subclass to reuse the code declared in a superclass, avoiding the duplication of common functionality between several classes. This reduces the amount of code that needs to be written and tested, and makes the code easier to understand and modify.

In this hierarchy, Student is the superclass, and `UndergraduateStudent` and `GraduateStudent` are its direct subclasses. `UndergraduateStudent` has four subclasses: `Freshman`, `Sophomore`, `Junior`, and `Senior`. `GraduateStudent` has two subclasses: `MastersStudent` and `DoctoralStudent`.

Each subclass inherits the properties and methods of its superclass. For example, `UndergraduateStudent` inherits the properties and methods of `Student`, and `Freshman` inherits the properties and methods of `UndergraduateStudent`. Subclasses can also override methods of their superclass to provide specialized behavior.

In this hierarchy, `Shape` is the superclass, and `TwoDimensionalShape` and `ThreeDimensionalShape` are its direct subclasses. `TwoDimensionalShape` has four subclasses: `Quadrilateral`, `Triangle`, `Circle`, and `Ellipse`. `Quadrilateral` has four subclasses: `Rectangle`, `Square`, `Parallelogram`, and `Trapezoid`. `ThreeDimensionalShape` has six subclasses: `Sphere`, `Cube`, `Cuboid`, `Cylinder`, `Cone`, and `Pyramid`.

Each subclass inherits the properties and methods of its superclass. For example, `TwoDimensionalShape` inherits the properties and methods of `Shape`, and `Rectangle` inherits the properties and methods of `Quadrilateral`. Subclasses can also override methods of their superclass to provide specialized behavior.

1. Protected and private are two access modifiers that control the visibility of the members of a class. Protected members are accessible within the class and its subclasses, while private members are only accessible within the class. The choice of using protected or private access in superclasses depends on the design goals and trade-offs of the software. Here are some points to consider:

Encapsulation: Encapsulation is the principle of hiding the implementation details of a class from the outside world. Private access supports a higher level of encapsulation and data hiding, as it prevents subclasses from accessing or modifying the internal state of the superclass. Protected access, on the other hand, exposes some of the superclass’s details to its subclasses, which may break the encapsulation and introduce coupling between the classes.

Reusability: Reusability is the ability to use existing code for different purposes or contexts. Protected access facilitates code reusability, as it allows subclasses to inherit and reuse the functionality of the superclass. Private access, however, restricts the reusability of the superclass’s code, as it forces subclasses to either duplicate or reimplement the functionality that is hidden in the superclass.

1. The code super.earnings() calls the earnings method of the superclass of the current subclass. This is useful when the subclass wants to use or modify the behavior of the superclass method in its own implementation. For example, if the subclass is CommissionEmployee and the superclass is Employee, then super.earnings() would return the base salary of the employee, which the subclass can then add the commission to.

The annotation @Override indicates that the following method is overriding a method from its superclass or an interface. This helps the compiler to check for errors and the reader to understand the intention of the method. For example, if the subclass is Dog and the superclass is Animal, then @Override before the makeSound method would mean that the subclass is providing its own definition of how a dog makes sound, different from the general animal sound.

The code super(firstArgument, secondArgument); invokes the constructor of the superclass with the given arguments. This must be the first statement in the constructor of the subclass, as it initializes the state of the superclass object before the subclass object. For example, if the subclass is Circle and the superclass is Shape, then super(color, filled); would call the constructor of Shape with the color and filled parameters, which are common attributes for all shapes.

1. inheritance: Here's how you can perform each of the taskS:

1. Specify that class `PieceWorker` inherits from class `Employee`:

public class PieceWorker extends Employee {

// Class body

}

This code declares the `PieceWorker` class and indicates that it is a subclass of the `Employee` class using the `extends` keyword.

2. Call superclass Employee's toString method from subclass PieceWorker's toString method:

@Override

public String toString() {

String employeeString = super.toString();

// Additional code specific to PieceWorker class

return employeeString;

}

This code overrides the toString method in the PieceWorker class and calls the superclass Employee's toString method using the super keyword. You can add any additional code specific to the PieceWorker class before returning the combined string.

3. Call superclass Employee's constructor from subclass PieceWorker's constructor:

public PieceWorker(String firstName, String lastName, String ssn) {

super(firstName, lastName, ssn);

// Additional code specific to PieceWorker class

}

This code declares a constructor for the PieceWorker class that takes three String parameters representing the first name, last name, and social security number. It calls the superclass Employee's constructor using the `super` keyword, passing the parameters. You can add any additional code specific to the PieceWorker class after the call to the superclass constructor.

1. In Java, when a subclass is created, it automatically inherits all the fields and methods from its superclass. However, there are cases where the subclass needs to perform additional initialization or customization beyond what is provided by the superclass.

In these cases, you can use the `super` keyword to invoke the superclass's constructor. By doing so, you are essentially calling the superclass's constructor to initialize the inherited fields and perform any necessary setup before the subclass-specific code is executed.

By using `super` as the first statement in a subclass constructor's body, you ensure that the superclass's constructor is called before any additional initialization or customization is performed in the subclass. This allows for proper initialization of the inherited fields and establishes the correct state of the object.

1. In Java, when a subclass defines an instance method with the same name as an instance method in its superclass, it overrides the superclass method. However, there may be cases where you want to invoke the superclass's method from within the subclass's method, instead of completely overriding it. In such cases, you can use the `super` keyword to refer to the superclass and access its version of the overridden method. By using `super` in the body of a subclass's instance method, you can call the superclass's method and execute its functionality, while still retaining the ability to add or modify behavior in the subclass's method.