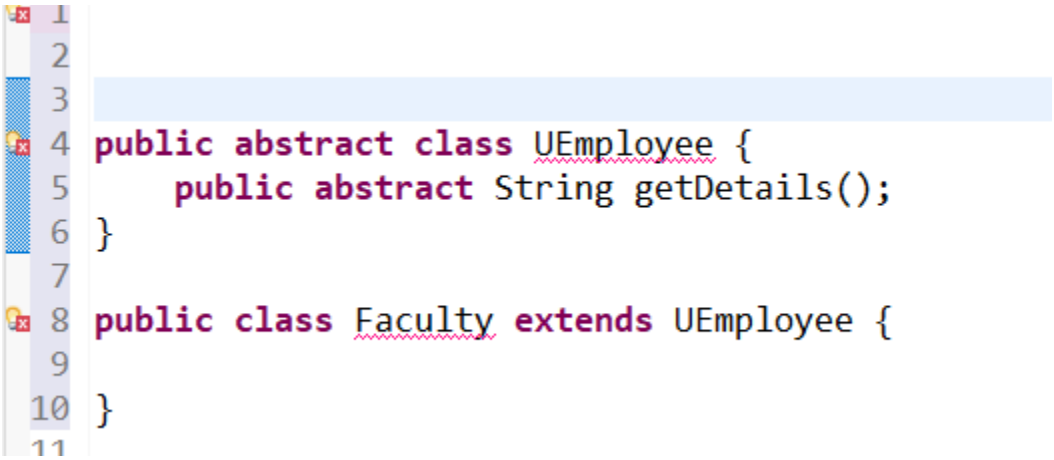
Credit Name: Chapter 8

Assignment Name: UEmployee

Describe the errors you've encountered while working on this assignment. What caused the error and how do you overcome the error?

1. Missing Abstract Method Implementation



```
public abstract class UEmployee {
    public abstract String getDetails();
}

public class Faculty extends UEmployee {

}
```

```
public class Faculty extends UEmployee {
    @Override
    public String getDetails() {
        return "Faculty Details";
    }
}
```

Error: The subclasses (Faculty and Staff) did not implement the getDetails() method from the abstract class UEmployee.

Cause: Abstract methods must be implemented by all non-abstract subclasses.

Fix: Add the getDetails() method to the subclasses with the appropriate implementation.

2. Incorrect Method Override

```
public abstract class UEmployee {
    public abstract String getDetails();
}

public class Faculty extends UEmployee {
    public String getFacultyDetails() {
        return "Faculty Details";
    }
}
```

```
public class Faculty extends UEmployee {
 @Override
 public String getDetails() {
     return "Faculty Details";
 }
}
```

Error: The method in the subclass had a different name (getFacultyDetails()) instead of getDetails().

Cause: Method names must match exactly when overriding abstract methods.

Fix: Rename the method in the subclass to getDetails() to ensure proper overriding.

3. Forgetting the @Override Annotation

```
public class Faculty extends UEmployee {
    public String getDetails() {
        return "Faculty Details";
    }
}
```

```
public class Faculty extends UEmployee {
 @Override
 public String getDetails() {
     return "Faculty Details";
 }
}
```

Error: The @Override annotation was missing, leading to potential mismatches in method signatures.

Cause: Without @Override, the compiler does not verify if the method correctly overrides a method in the superclass.

Fix: Add the @Override annotation to ensure the method signature matches the abstract method in the parent class.

4. Using abstract with Concrete Methods

```
public abstract class UEmployee {
    public abstract void getName() {
        System.out.println("Name");
    }
}
```

```
public abstract class UEmployee {
    public abstract void getName();
}
```

Error: An abstract method in UEmployee had a body, which is not allowed.

Cause: Abstract methods cannot have a body—they must be implemented in the subclasses.

Fix: Remove the method body or remove the abstract keyword to make it a concrete method.

5 . Missing Constructor in Subclasses

```
public abstract class UEmployee {
    public UEmployee(String name) {}
}

public class Faculty extends UEmployee {

}
```

```
public class Faculty extends UEmployee {
    public Faculty(String name) {
        super(name);
    }
```

Error: The subclass Faculty did not have a constructor matching the UEmployee superclass constructor, causing a compilation error.

Cause: Java requires subclasses to explicitly call the superclass constructor if one is defined.

Fix: Add a constructor in the subclass and use super() to call the superclass constructor.

7. Using private Instead of protected

```
public abstract class UEmployee {
    private String name;
}
```

```
public abstract class UEmployee {
    protected String name;
}
```

Error: Declaring the name and salary fields in UEmployee as private prevented access in the subclasses.

Cause: The private access modifier restricts access to the class in which it is defined.

Fix: Use protected to allow access in the subclasses.

8. Forgetting to Call super()

```
public abstract class UEmployee {
    public UEmployee(String name) {}
}

public class Faculty extends UEmployee {
    public Faculty(String name) {}
}
```

Error: The subclass Faculty did not call the superclass constructor, causing a compilation error.

Cause: Java requires a call to super() in a subclass constructor when the superclass defines a constructor.

Fix: Add super() in the subclass constructor to call the parent class constructor.