AI generated solutions verified by @markfmyt (subject to being wrong)

5. The details of the `Undergraduate` object `u1` are printed using the `toString()` method. The state that is printed comes from the `Student` superclass, as `Undergraduate` is a subclass of `Student` and inherits all public and protected members (fields and methods) from `Student`.

8. If the `toString()` method is not overridden in the `Undergraduate` class, it will not print the new attributes (`minor`, `major`, `credits`) added in `Undergraduate`. It will only print the attributes inherited from the `Student` class. This is because the `toString()` method of the `Student` class is unaware of the new attributes added in the `Undergraduate` class.

10. After refining the `toString()` method in the `Undergraduate` class, all the state of the `Undergraduate` objects (`u1`, `u2`, `u3`) should print properly. For the `Student` objects (`s1`, `s2`), their state will be printed as before, because their `toString()` method has not been changed.

Bonus Question: If you comment off both of the `toString()` methods in the `Undergraduate` and `Student` classes, the code still works because every class in Java is a subclass of the `Object` class, which has a `toString()` method. The output will be the class name, followed by an '@' sign, followed by the unsigned hexadecimal representation of the hash code of the object. This is not very informative, but it is a valid default.

12. If you overload the constructor of the `Student` class, you do not have to change the code for the `Student` objects `s1` and `s2` in the `StudentApp` class, because overloading a constructor (or any method) means providing a new constructor with a different parameter list. The original constructor is still available for use.

13. When you compile your `Undergraduate` class, the no-argument constructor of the `Student` superclass is called by the `Undergraduate` constructor, because in Java, if a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

14. If you comment off the no-argument constructor in the `Student` class, you will get a compilation error in the `Undergraduate` class, because the Java compiler tries to insert a call to the no-argument constructor of the superclass, but it does not exist. This can be fixed by providing a no-argument constructor in the `Student` class, or by explicitly calling a different superclass constructor in the `Undergraduate` constructor.

20. After changing the status of the `Undergraduate` and `Postgraduate` objects and calculating their fees, you should observe that the fees are calculated correctly according to the new status and the number of credits. The details printed should also reflect the changes in status and fees.

21. When you invoke the `calculateFees()` method on the `Student` objects `s1` and `s2`, it will calculate the fees based on the implementation of the method in the `Student` class, because the method has not been overridden in the `Student` class. If the method is not defined in the `Student` class, you will get a compilation error. If it is defined, it will calculate the fees based on its implementation, which may be different from the implementations in the `Undergraduate` and `Postgraduate` classes.