Lab Assignment: Inheritance Inheritance

You will be given these incomplete class structures:

1. `Punnet.java (this one is complete)`
2. `LivingThing.java`
3. `Animal.java (inherits from LivingThing)`
4. `Dog.java (inherits from Animal)`
5. `SomethingNew.java (you decide what this inherits from)`
6. `Gene.java`

Your task is to complete the classes so that the simulation runs properly. In addition to the class notes on the basics of graphics applications, you should look up the APIs for `Color` (https://docs.oracle.com/javase/7/docs/api/index.html?java/awt/Color.html) and `Graphics` (https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html). These will help you to draw interesting things with interesting colors.

If you finish the lab, there is an extra credit opportunity. However, please make separate versions of your files so that you do not lose your work from the main part of the lab.

Extra Credit
An Abstract Class is one that is never intended to be constructed. It is there merely for reference type. In it, you will find at least one abstract method, which is nothing more than a method signature. Every class that inherits from an abstract class *must* implement the abstract method, otherwise a compile time error results.

1. Study the abstract class version of `Gene.java` and find the abstract method.
2. Research incomplete dominance and codominance and how they differ from simple dominance.
3. Create three new classes that inherit from `Gene`.
    a. `SimpleGene`: This gene should implement simple dominance. You will need to implement the method *dominant* and *mix* (this should be short and should call `standardMix`). This method should return a 1 if the gene is dominant and 0 otherwise. Does anything else need overridden?
    b. `IncompleteGene`: This gene should implement incomplete dominance. The two characters should be one-digit `int`s in character form, although you should have a constructor that accounts for two actual `int`s being sent in. You should also have a private data `int` called value. The value is calculated by selecting a random `int` between the two character `int`s. When you mix two genes, you should still select one component from `parent1` and one component from `parent2`. `dominant` should return value.
    c. `CoGene`: This gene should implement co-dominance. The mixing will work the same as the previous two. `dominant` should return a 0 for homozygous recessive (e.g., bb), 1 for heterozygous (e.g., Bb or bB), and 2 for a homozygous dominant pair (e.g., BB).
4. You will need to modify the `LivingThing` class slightly. The body color gene should be made a `CoGene` although the reference type should remain a `Gene`. What does this mean? Half and half? Polka dots? Stripes? You decide. You will need to change the `drawMe` method to reflect this dominance.
5. In the animal class, the eye color gene should remain a `SimpleGene`. What other changes need to be made?
6. The tail gene for the dog should be an `IncompleteGene`. You will need to make changes to the `drawMe` method to reflect this dominance.

7. Finally, do you need to make any changes to the `Punnett` class? Have a text field that allows the user to input an integer for the size gene. (Look up the `TextField` API.) Do any other changes need to be made? You decide.