Replace Conditional with Polymorphism

Problem

You have a conditional that performs various actions depending on object type or properties.

```
class Bird {
  //...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

Solution

Create subclasses matching the branches of the conditional. In them, create a shared method and move code from the corresponding branch of the conditional to it. Then replace the conditional with the relevant method call. The result is that the proper implementation will be attained via polymorphism depending on the object class.

```
abstract class Bird {
  //...
  abstract double getSpeed();
}

class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);
  }
}

// Somewhere in client code
speed = bird.getSpeed();
```

Why Refactor

This refactoring technique can help if your code contains operators performing various tasks that vary based on:

- Class of the object or interface that it implements
- Value of an object's field
- Result of calling one of an object's methods

If a new object property or type appears, you will need to search for and add code in all similar conditionals. Thus the benefit of this technique is multiplied if there are multiple conditionals scattered throughout all of an object's methods.

Benefits

- This technique adheres to the *Tell-Don't-Ask* principle: instead of asking an object about its state and then performing actions based on this, it is much easier to simply tell the object what it needs to do and let it decide for itself how to do that.
- Removes duplicate code. You get rid of many almost identical conditionals.
- If you need to add a new execution variant, all you need to do is add a new subclass without touching the existing code (*Open/Closed Principle*).

How to Refactor

Preparing to Refactor

For this refactoring technique, you should have a ready hierarchy of classes that will contain alternative behaviors. If you do not have a hierarchy like this, create one. Other techniques will help to make this happen:

- Replace Type Code with Subclasses. Subclasses will be created for all values of a particular object property. This approach is simple but less flexible since you cannot create subclasses for the other properties of the object.
- Replace Type Code with State/Strategy. A class will be dedicated for a particular object property and subclasses will be created from it for each value of the property. The current class will contain references to the objects of this type and delegate execution to them.

The following steps assume that you have already created the hierarchy.

Refactoring Steps

1. If the conditional is in a method that performs other actions as well, perform Extract Method.
2. For each hierarchy subclass, redefine the method that contains the conditional and copy the code of the corresponding conditional branch to that location.
3. Delete this branch from the conditional.
4. Repeat replacement until the conditional is empty. Then delete the conditional and declare the method abstract.