Tutorial 5 - More Behavioural Design Patterns

1) Using the State Design Pattern, rewrite the following class so that the conditional statement (if .. else..) is not required:

```
class Engine {

    boolean started = false;

    void setState(boolean b) {started = b;}

    void printState() {
       if (started)
          System.out.println("Engine is Started");

      else
        System.out.println("Engine is Stopped");
    }
 }
```

2) In the software that performs engine management, the objects responsible for ignition control, fuel metering and exhaust emission monitoring need to know when the engine is started or stopped. Suggest a design pattern that might be used to organise this interaction and modify the Engine class to support the interaction.

3) What design pattern is being used in the following code fragment:

```
abstract class Starter {

  public void start() {
     if (!battery()) return;
     if (!fuel()) return;
     if (!neutral())return;
         (Engine.getEngine()).start();
  }

  abstract boolean battery();
  abstract boolean fuel();
  abstract boolean neutral();
}
```

4) Suggest a Design Pattern that might be used to allow the flexibility to add new pre-conditions on starting the engine. Outline the Java code that could be used.