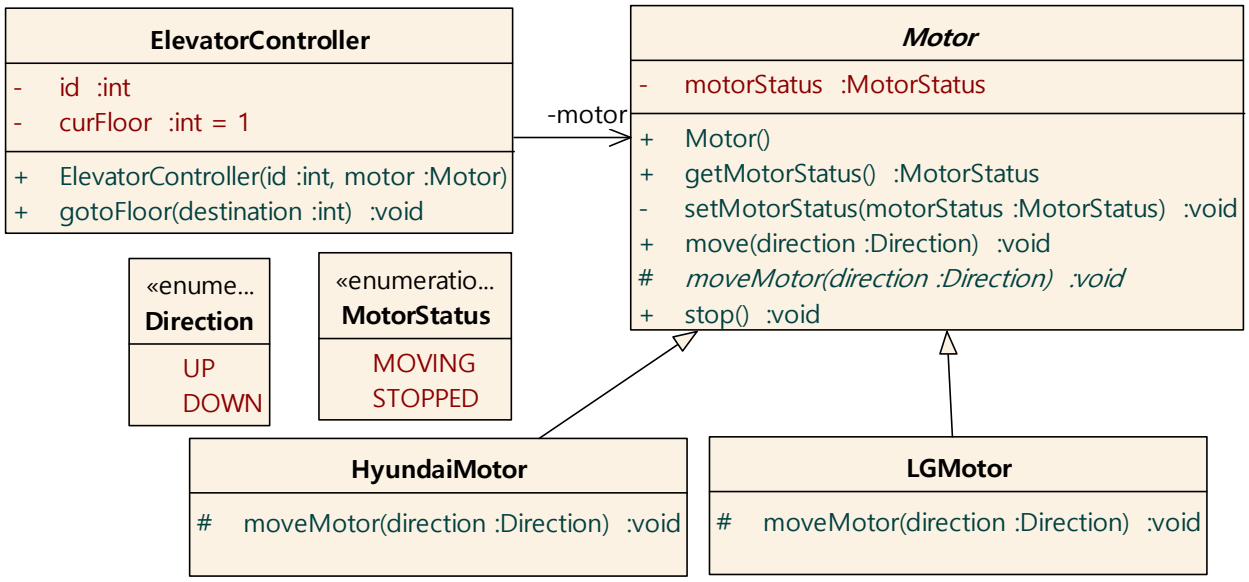
Factory Method pattern

PRACTICE – MOTOR FACTORY

Motors

- Elevator needs to support two kinds of motors: HyundaiMotor and LGMotor.
- Both motor classes have many common codes which are captured in Motor



Source Code – Motor

```
public abstract class Motor {
  private Door door ;
  private MotorStatus motorStatus ;
  public Motor(Door door) {
    this.door = door ; motorStatus = MotorStatus.STOPPED ;
  }
  public MotorStatus getMotorStatus() { return motorStatus; }
  private void setMotorStatus(MotorStatus motorStatus) {
    this.motorStatus = motorStatus;
  }
  public void move(Direction direction) {
    MotorStatus motorStatus = getMotorStatus() ;
    if (  motorStatus == MotorStatus.MOVING ) return ;
    DoorStatus doorStatus = door.getDoorStatus() ;
    if ( doorStatus == DoorStatus.OPENED ) door.close() ;
    moveMotor(direction) ;
    setMotorStatus(MotorStatus.MOVING) ;
  }
  protected abstract void moveMotor(Direction direction) ;
}
```

Implements skeleton of an algorithm

This step is deferred to subclasses

Source Code – Concrete Motors

- Concrete motor implements the deferred step to perform subclass-specific steps of the algorithm

```
public class HyundaiMotor extends Motor {
  public HyundaiMotor(Door door) {
    super(door) ;
  }
  protected void moveMotor(Direction direction) {
    System.out.println("Hyundai Motor: Move " + direction) ;
  }
}
```

```
public class LGMotor extends Motor {
  public LGMotor(Door door) {
    super(door) ;
  }
  protected void moveMotor(Direction direction) {
    System.out.println("LG Motor: Move " + direction) ;
  }
}
```


Source Code – Elevator Controller

```
public class ElevatorController {
  private int id ;
  private int curFloor = 1 ;
  private Motor motor ;
  public ElevatorController(int id, Motor motor) {
    this.id = id ; this.motor = motor ;
  }
  public void gotoFloor(int destination) {
    if ( destination == curFloor ) return ;
    Direction direction ;
    if ( destination > curFloor ) direction = Direction.UP ;
    else direction = Direction.DOWN ;
    motor.move(direction) ;
    System.out.print("Elevator [" + id + "] Floor: " + curFloor) ;
    curFloor = destination ;
    System.out.println(" ==> " + curFloor + " with " +
      motor.getClass().getSimpleName() ) ;
    motor.stop() ;
  }
}
```

Source Code: Client

```
public class Client {
 public static void main(String[] args) {
  Motor lgMotor = new LGMotor() ;
  ElevatorController controller1 = new ElevatorController(1, lgMotor) ;
  controller1.gotoFloor(5) ;
  controller1.gotoFloor(3) ;

  Motor hyundaiMotor = new HyundaiMotor() ;
  ElevatorController controller2 = new ElevatorController(2, hyundaiMotor) ;
  controller2.gotoFloor(4) ;
  controller2.gotoFloor(6) ;
 }
}
```

```
move LG Motor UP
Elevator [1] Floor: 1 ==> 5 with LGMotor
move LG Motor DOWN
Elevator [1] Floor: 5 ==> 3 with LGMotor
move Hyundai Motor UP
Elevator [2] Floor: 1 ==> 4 with HyundaiMotor
move Hyundai Motor UP
Elevator [2] Floor: 4 ==> 6 with HyundaiMotor
```


Rewrite the Client Code with MotorFactory

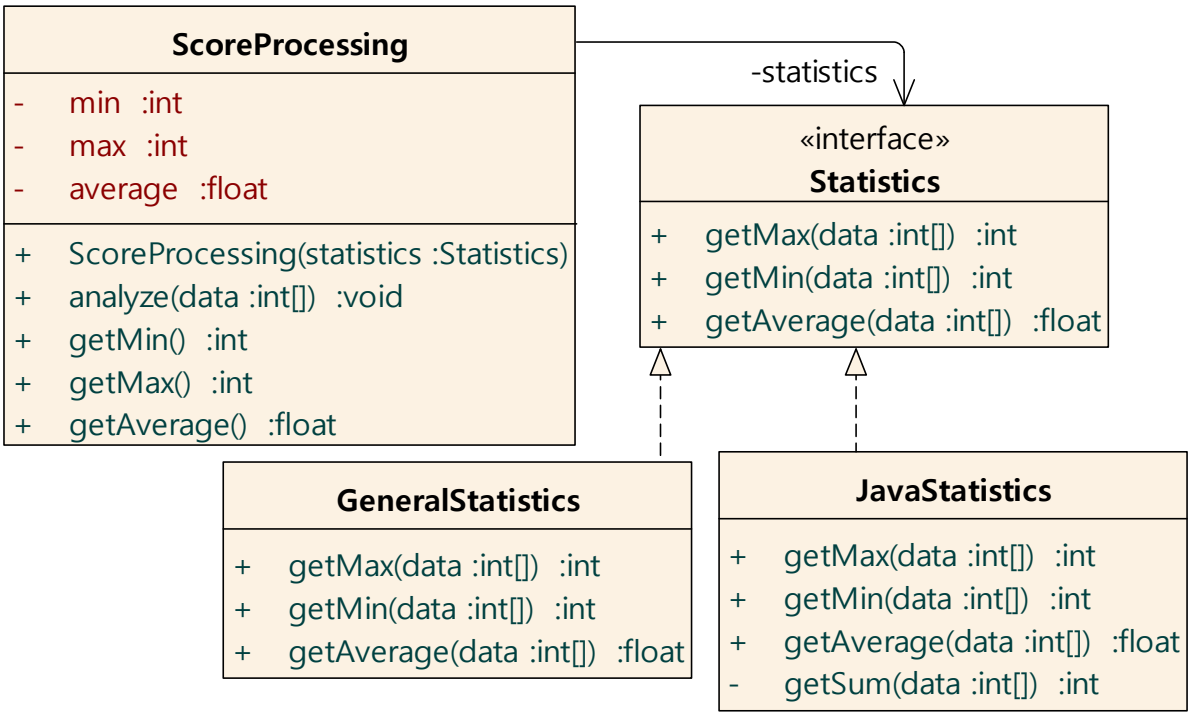
Source Code: MotorFactory

Source Code: Client

Factory Method and other Patterns

- Factory Method pattern can create objects with common ancestor
- Thus, Factory Method pattern is usually used with patterns based on inheritance hierarchy
- For example, Strategy and State

Factory Method and Strategy



Factory Method and Strategy

```
public enum StatID { GENERAL, JAVA }
```

```
public class StatisticsFactory {
  public static Statistics getStatistics(StatID statID) {
    Statistics state = null ;
    switch ( statID ) {
      case GENERAL : state = new GeneralStatistics() ; break ;
      case JAVA : state = new JavaStatistics() ; break ;
    }
    return state ;
  }
}
```

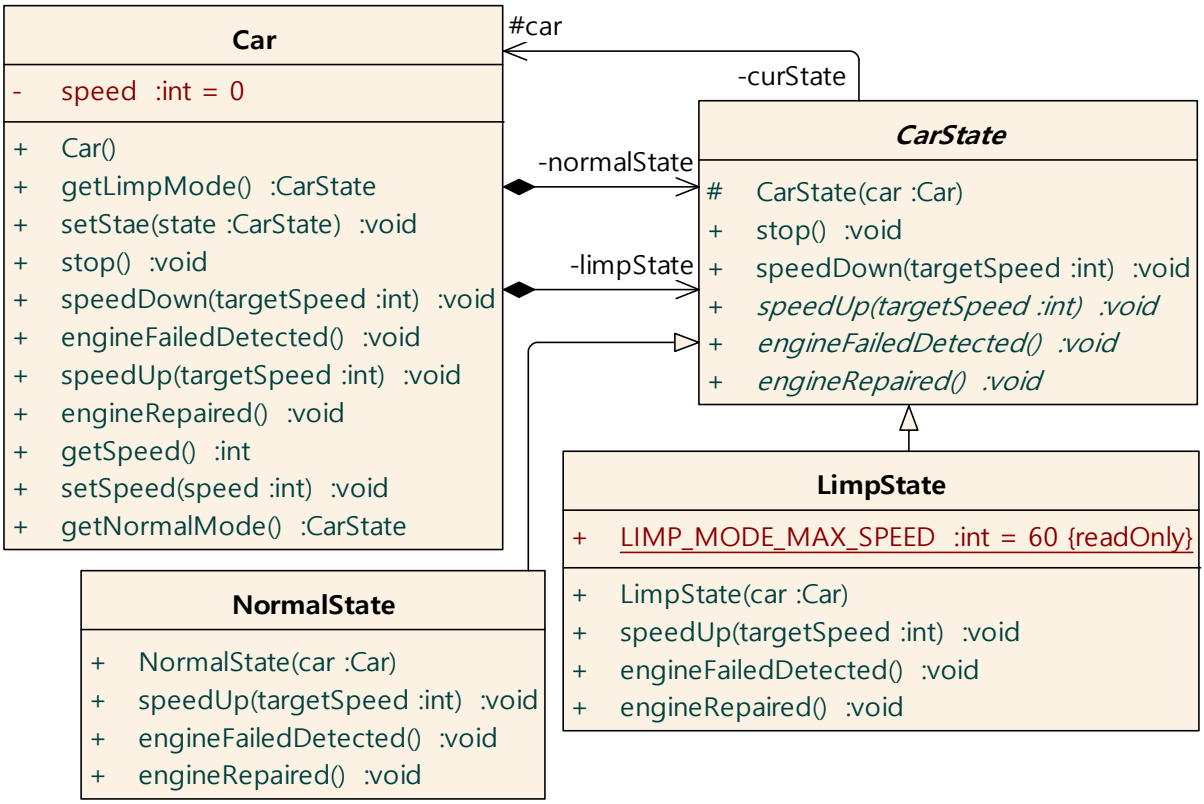

Factory Method and Strategy

```
int[] data = {0, 50, 10, 30, 70} ;

Statistics generalStatistics =
  StatisticsFactory.getStatistics(StatID.GENERAL) ;
ScoreProcessing proc1 = new ScoreProcessing(generalStatistics) ;
proc1.analyze(data) ;


Statistics javaStatistics =
  StatisticsFactory.getStatistics(StatID.JAVA) ;
ScoreProcessing proc2 = new ScoreProcessing(javaStatistics) ;
proc2.analyze(data) ;
```

Factory Method and State



Factory Method and State

```
public enum CarStateID { NORMAL, LIMP_MODE }
```

```
public class CarStateFactory {
  public static CarState getState(CarStateID stateID, Car car) {
    CarState state = null ;
    switch ( stateID ) {
      case NORMAL : state = new NormalState(car) ; break ;
      case LIMP_MODE : state = new LimpState(car) ; break ;
    }
    return state ;
  }
}
```

Factory Method and State

```
public class Car {
  private int speed ;
  private CarState normalState ;
  private CarState limpState ;
  private CarState curState ;

  public Car() {
    normalState = CarStateFactory.getState(CarStateID.NORMAL, this) ;
    limpState = CarStateFactory.getState(CarStateID.LIMP_MODE, this) ;
    curState = normalState ;
  }
```