*public interface MotorVehicle* {

 *void* build();

}

*public class* Car *implements MotorVehicle* {
 @Override
 *public void* build() {
 System.*out*.println("Build Car");
 }
}

*public class* Motorcycle *implements MotorVehicle* {
 @Override
 *public void* build() {
 System.*out*.println("Build Motorcycle");
 }
}

*public abstract class* MotorVehicleFactory {
 *public MotorVehicle* create() {
 *MotorVehicle* vehicle = createMotorVehicle();
 vehicle.build();
 *return* vehicle;
 }
 *protected abstract MotorVehicle* createMotorVehicle();
}

*public abstract class* MotorVehicleFactory {
 *public MotorVehicle* create() {
 *MotorVehicle* vehicle = createMotorVehicle(); *// I will create the object for you* vehicle.build(); *// I will build the object for you
 return* vehicle; *// I will return the object to you
 // these tasks are predesigned and are common to all vehicles
 // all these are encapsulated* }
 *protected abstract MotorVehicle* createMotorVehicle();
}

*public class* CarFactory *extends* MotorVehicleFactory {
 @Override
 *protected MotorVehicle* createMotorVehicle() {
 *return new* Car();
 }
}

*public class* MotorcycleFactory *extends* MotorVehicleFactory {
 @Override
 *protected MotorVehicle* createMotorVehicle() {
 *return new* Motorcycle();
 }
}

*public class* MotorVehicleFactoryTest {

 @Test
 *public void* givenCarFactory\_whenCreateMotorVehicle\_thenInstanceOf() {
 MotorVehicleFactory factory = *new* CarFactory();
 *MotorVehicle* car = factory.create();

 *assertThat*(car).isInstanceOf(*MotorVehicle*.*class*);
 *assertThat*(car).isInstanceOf(Car.*class*);
*// car.build();* }

 @Test
 *public void* givenMotorcycleFactory\_whenCreateMotorVehicle\_thenInstanceOf() {
 MotorVehicleFactory factory = *new* MotorcycleFactory();
 *MotorVehicle* motorcycle = factory.create();
 *assertThat*(motorcycle).isInstanceOf(*MotorVehicle*.*class*);
 *assertThat*(motorcycle).isInstanceOf(Motorcycle.*class*);
*// motorcycle.build();* }
}

****