**Question 1: Study the following piece of code from RentalContract.java. What does it do?**

**Code: RentalContract.java**

```
if (insuranceAdded) {
    insurance = ((Insurable)rentable).calculateInsurancePremium(days);
}
```

**This code block works in multiple steps:**

1. `if (insuranceAdded)`: This line will check whether the insurance has been added or not to begin the calculation of the insurance rate.
2. `insurance = ((Insurable) rentable)`: This line will cast the type insurable to the rentable object in question rentable, which is in type Rentable before this. This allows us to call Insurable methods on it. This is necessary because the calculation of the insurance premium is not possible until the object in question is Insurable, as Rentable does not contain the method.
3. `calculateInsurancePremium(days)`: This method can be called no matter the type of vehicle as long as it is Rentable and able to be converted to Insurable (Both of these are present in the Vehicle class so it should be possible). Based upon the object involved, Car method will override the Vehicle method, Truck will also do the same, but if it is neither then the generic Vehicle calculation method is used.

**Question 2: Can you override getBaseInsuranceRate in Car? Why or why not?**

**Code: Vehicle.java**

```
public final double getBaseInsuranceRate() {
    return 20.0;
}
```

You cannot override it. The reason is that it is a final method. This is frozen and will remain the same for all vehicles. Attempting to override it will cause an error.

**Question 3: Use calculateRentalPrice(days, gpsIncluded) on various instances**

**Code: Vehicle.java,**

```
//Rentable.java
double calculateRentalPrice(int days);
//Truck.java
public double calculateRentalPrice(int days) {
    double dailyRate = 90 + (10 * loadCapacityInTons);
    return dailyRate * days;
}
//Car.java Overloaded Method
public double calculateRentalPrice(int days, boolean gpsIncluded) {
    double base = calculateRentalPrice(days);
    if (gpsIncluded) {
        base += 10.0 * days; // GPS daily fee
    }
    return base;
}
```

Looking at the methods available for Rentable, Truck, and Car classes the only one which is able to compile correctly with the gpsIncluded parameter included is the Car class. The rest of the examples fail to execute properly.

### Question 4: Behavior of printVehicleType with different object types

**Code: Vehicle.java**

```
//Vehicle.java
public static void printVehicleType() {
System.out.println("Generic Vehicle"); }
//Truck.java
public static void printVehicleType() {
System.out.println("Truck"); }
//Car.java
public static void printVehicleType() {
System.out.println("Car"); }
```

These methods are special as they are defined as static. Since they are static they cannot be overridden and so they are hidden. When calling a hidden method the type of the reference variable is checked for which method to use rather than the type of the object the method is called on. This means that if the lines `Vehicle v = new Car(...);` `v.printVehicleType();` are called then the result will be “Generic Vehicle” instead of “Car”. This happens because static methods belong to classes and not instances.