1. Write a class Car with integer variables viz. yearOfMake & string variables engineNumber & type(like Sedan, Hatchback & SUV). Create proper getter & setter methods for these variables with proper access specifiers. Now create subclasses (Swift, XUV, SCross) for Car with integer variables viz. seats, airbags & a string variable model, color. Create proper getter & setter methods for these variables in the subclasses.
2. From the runner class create an instance for Swift. Using that, call all the setter methods & then all the getter methods in Swift class alone. Print the values returned from the getter methods
3. From the runner class create an instance for SCross. Using that, call all the setter methods & then all the getter methods in SCross including superclass methods. Print the values returned from the getter methods
4. In the runner class create a method which accepts a Car object as an argument.Try to invoke that method with objects of XUV, SCross & Swift. For eg Swift obj = new Swift()

5. In the method created above, identify the actual underlying object of the incoming Car object. If the incoming object is Swift print “Hatch”, if XUV print SUV & if SCross print Sedan.
6. In the runner class create another method which accepts a Swift object as an argument.
    1. Try to invoke that method with an Swift object i.e Swift obj = new Swift()
    2. Try to invoke that method with a Car object which is assigned a Swift Object. i.e Car obj = new Swift(); Understand how different this scenario is from (i).
    3. Try to invoke that method with objects of XUV, SCross & understand how different this scenario is from the 4th task.
7. In the Car class create a method maintenance () . Inside that print " Car under maintenance" . Now override this method in SCross class & print “Maruti SCross under maintenance” .

   In the runner class perform the following & understand the flow in each scenario.
    1. Create an instance for Scross object & call maintenance() method
    2. Create an instance for SCross object & assign it to Car variable(Car obj = new SCross();) & call maintenance() method
    3. Create an instance for Car object & call maintenance() method
    4. Create an instance for Swift object & call maintenance() method.
8. In the Car class create an overloaded constructor which accepts a String & print the incoming String. In the XUV class’s default constructor, try to call the overloaded Constructor of

the super class. You should not create the overloaded constructor in XUV class. Now from the runner class,

1. Try to create an instance of XUV using default constructor
2. Try to create an instance of XUV using the overloaded constructor

9. Create an abstract class BirdAbstract with two methods: fly() & speak(). Both the fly() & speak() can print a statement & these methods need not be abstract. From a runner class try to create an instance for the BirdAbstract class.
    1. Now compile the runner class & check the output & understand.
    2. Now create a subclass ParrotMod extending BirdAbstract & you need not override the methods fly & speak. Create an instance for the ParrotMod from the runner class & invoke the fly & speak methods.
10. Create a class Bird with two methods: fly() & speak(). Let the fly() method alone be abstract. Now create a subclass Duck extending Bird & provide implementation for the fly. Create an instance for the Duck from the runner class & invoke the fly & speak methods.

Abstract classes can have constructors even though an object can't be instantiated for it. It's used to initialize variable values for the subclasses deriving it.