**Create separate packages for all the tasks. You do not need to create separate projects.**

Task 1

* Create a package, name: com.itacademy.java.oop.basics.task1
* Create class ShapeApplication with main method.
* Create abstract class Shape with abstract methods calculateArea(), calculatePerimeter().
* Create class Square with field side of double type.
* Create class Circle with field radius of double type.
* Circle and Square must extend Shape and implement its methods.

Requirements:

* Create object for Circle and Square.
* Use implemented methods to calculate area and perimeter for both objects.
* Print out response implementing toString()

Task 2

* Create a package, name: com.itacademy.java.oop.basics.task2
* Create class BicycleApplication with main method.
* Create interface Bicycle with methods changeGear(int newGear): void, speedUp(int increment): void (should increase bicycle speed), applyBrakes():void (should decrease bicycle speed).
* Create classes MountainBike and RoadBike with fields int gear and int speed. They must implement Bicycle interface.
* Create enum Bike with values mountain and road.
* Create a class Speedometer with two static methods chooseWinner(MountainBike mountainBike, RoadBike roadBike): Bike and chooseWinner(int mountainBikeSpeed, int roadBikeSpeed): Bike
* Use all the methods and play around to see if all of the business logic below works as expected.

Requirements:

* Create one MountainBike and one RoadBike.
* MountainBike applyBrakes() – should allow bicycle to move backwards (negative speed) as fast as -10. If braking value decreases speed to -11 or lower, it should not allow to brake and print out a message saying that value is too high and suggest the highest possible value to use to brake a mountain bike. Braking value cannot be negative number. If number is negative print a message that it is negative and suggest using positive numbers. If everything passes mountain bike’s speed should be successfully decreased. Current speed must be printed after it was changed.
* MountainBike speedUp() – should allow only positive numbers. If negative numbers are supplied print a message saying it is negative and suggest using positive numbers. Mountain bike can achieve maximum speed of 100. If increasing, its speed makes it over 100 do not increase speed and print a message saying that speed is too high and suggest the highest possible increment it can be increased. If everything passes increase mountain bike speed and print, it’s new speed.
* MountainBike changeGear() – gear can only be increased or decreased by only 1 or -1. If anything, other than 1 or -1 is supplied print a message saying that value is not allowed and suggest using 1 or -1. Mountain bike gear can not be negative. Lowest allowed gear is 0 highest allowed gear is 20. If gear change brakes these limits print a message saying that it was either over the max 20 or below 0. If everything passes print out new mountain bike gear.
* RoadBike applyBrakes() – it should not allow road bike to move backwards. It’s speed cannot be negative. Brakes can only be applied using negative values. If braking using positive value print a message suggesting using negative value. Road bike speed cannot be negative. If everything passed print new road bike speed.
* RoadBike speedUp() – should allow only positive numbers. If negative numbers are supplied print a message saying it is negative and suggest using positive numbers. Road bike can achieve maximum speed of 50. If increasing, it’s speed makes it over 50 do not increase speed and print a message saying that speed is too high and suggest the highest possible increment it can be increased. If everything passes increase mountain bike speed and print, it’s new speed.
* RoadBike changeGear() – gear can only be increased or decreased by only 1 or 2 or -1 or -2. If anything, other than 1 or 2 or -1 or -2is supplied print a message saying that value is not allowed and suggest using the closest allowed value in regards to the supplied value. Road bike gear cannot be negative. Lowest allowed gear is 0 highest allowed gear is 10. If gear change brakes these limits print a message saying that it was either over the max 10 or below 0. If everything passes print out new road bike gear.
* Speedometer chooseWinner() – it should return enum value represting a faster bike. It should compare speeds using just the speed values or using objects to get the speed values and comparing them. If mountain bike is faster mountain value should be returned and vice versa.

Task 3

* Create a package, name: com.itacademy.java.oop.basics.task3
* Create class AtmApplication with main method.
* Create abstract class Card with methods credit(double) and debit(double amount) fields balance(double), cardHolderName(string) and cardNumber(string).
* Create class CreditCard with fields interest(double), credit(double) which extends Card.
* Create class DebitCard and extend Card.
* Create interface Atm with method to withdraw using CreditCard and DebitCard.
* Create class MyBankAtm implementing Atm interface.
* Use all the methods and play around to see if all of the business logic below works as expected.

Requirements:

* Create one CreditCard and one DebitCard.
* Use Immutability, Overriding, Overloading.
* Create one MyBankAtm. MyBankAtm must have it’s balance as a field. It will be used when withdrawing money from the ATM. You can not withdraw more money then there is in ATM. When trying to withdraw more money than there is in ATM throw an exception InsuficientAtmFundsException.
* CreditCard has an interest which is a percentage value. It is used when withdrawing money from ATM. For example, if withdrawing 100 and interest is 1% then it will deduct 1% from 100 from credit card balance as payment for withdrawal.
* DebitCard withdrawal does not cost anything it just increases card balance.
* CreditCard credit value is not modifiable after object was created.
* debit method works the same for both cards it reduces its balance. Debit card can’t have negative balance. Only credit card can have negative balance. Credit card negative balance can not be less than it’s credit value. If trying to debit credit card makes it more than it’s credit throw an exception NotEnoughtCreditException. If trying to debit Debit cards will result in having negative balance throw exception NotEnoughBalanceException
* debit methods works the same for both Debit and Credit cards it increases their balance.
* You must catch exceptions and display explanation messages in console what happened. Your custom exceptions must have their unique messages.
* When using Atm you must use debit and credit to change card balance values.