Oop

hot shocolate system

main

```
//import 'package:flutter/material.dart';
import 'package:hotshproject/hotshoc.dart';

void main() {
  print('n');
  //object
  var object1 = hotshoclate();
  // object1.makeorder(1);
  object1.hotshlevel = 100;
  object1.waterLevel = 500;
  object1.makeorder(1);
  print(object1.hotshlevel);
}
```

Hotshoc class

```
class hotshoclate{
  List hotshsize=[7,9,8];
  double waterLevel=1000;
  double hotshlevel=4000;


  void turnonoption(){
    print('option chosn');
  }
  void turnonoptioff(){
    print('option close');
  }

  bool iswaterenough(int hotshsize ){
    if(hotshsize==1&& waterLevel==500)
      {
        return true;
      }
    else{
    return false;}
  }
  bool ishotenough(int hotshsize){
    if(hotshsize==1&& hotshlevel==10)
    {
      return true;
    }
    else{
      return false;}
     //return true;
  }
  void warnhotshoclevellow(){
```

```
  }

  void makeorder(int hotshsize)
  {
    turnonoptioff();
    if(hotshsize==1){
      bool waterenough =iswaterenough( hotshsize);
      bool hotenouh=ishotenough( hotshsize);
      if (waterenough&&hotenouh)
        {
          //after make decress the water and leave the option button on
          waterLevel -=500;
          hotshlevel -=10;
          print('hoschready');
          turnonoption();
        }
      else{
        print('not enough');
      }

    }
  }
}
```

.....

Constructor

in class

```
hotshoclate({List ?l,double ? h,double ?s}){
  this.hotshsize=l!;
  this.waterLevel=h!;
  this.hotshlevel=s!;
}
```

in main

```
import 'package:hotshproject/hotshoc.dart';

void main() {
  print('n');
  //object
  List hotshsize=[1];
  //var object1 = hotshoclate( hotshsize,100,500);
  var object=hotshoclate(l:hotshsize,h:100,s: 500);
  // object1.makeorder(1);
  //object1.hotshlevel = 100;
 //object1.waterLevel = 500;
  //object1.makeorder(1);
  //print(object1.hotshlevel);
  print(object.hotshlevel);
```

```
}
```

.....

Encapsulation

Put _before variable then use set, get to get it

Main

```
//import 'package:flutter/material.dart';
import 'package:hotshproject/hotshoc.dart';

void main() {
  print('n');
  //object
  List hotshsize=[1];
  //var object1 = hotshoclate( hotshsize,100,500);
  var object=hotshoclate(l:hotshsize,h:50,s: 10);









  // object1.makeorder(1);
  //object1.hotshlevel
  // = 100;
  //object1.waterLevel = 500;
  //object1.makeorder(1);
  //print(object1.hotshlevel);
  print(object.hotshlevel);
  object.makeorder(1);

//////////
  //set
  object.numoption=3;
  //get
  print(object.numoption);

}
```

Hotschcclass

```
class hotshoclate{

  //encapsolation
  //private
  int _numoption=2;
```

```
  List hotshsize=[7,9,8];
  double waterLevel=1000;
  double hotshlevel=4000;
//constructor
  hotshoclate({List ?l,double ? h,double ?s}){
    this.hotshsize=l!;
    this.waterLevel=h!;
    this.hotshlevel=s!;
  }

  //set is like void function ,get like int type function

  set numoption(int numoption){
    if(numoption<=3){this._numoption=numoption;}
  }

   int get numoption{
    return this._numoption;
   }

 //#methods
  void turnonoption(){
    print('option chosn');
  }
  void turnonoptioff(){
    print('option close');
  }

  bool iswaterenough(int hotshsize ){
    if(hotshsize==1&& waterLevel==500)
      {
        return true;
      }
    else{
    return false;}
  }
  bool ishotenough(int hotshsize){
    if(hotshsize==1&& hotshlevel==10)
    {
      return true;
    }
    else{
      return false;}
     //return true;
  }
  void warnhotshoclevellow(){

  }

  void makeorder(int hotshsize)
  {
    turnonoptioff();
    if(hotshsize==1){
      bool waterenough =iswaterenough( hotshsize);
      bool hotenouh=ishotenough( hotshsize);
```

```
      if (waterenough&&hotenouh)
        {
          //after make decress the water and leave the option button on
          waterLevel -=500;
          hotshlevel -=10;
          print('hoschready');
          turnonoption();
        }
      else{
        print('not enough');
      }

    }
  }
}
```

.....

train system

inheritance with extends

polymorphism with override methods,

main

```
import 'package:untitledtr/retrain.dart';
import 'package:untitledtr/seat.dart';
import 'package:untitledtr/train.dart';

void main() {
  final List<Seat>b=[Seat(type: "rest",price: "50pound")];
var n1=ReTrain(seats: b);
n1.id="6";
n1.seats=b;
print(n1.id);
//print(n1.seats);
//call polymorph
Train bb= ReTrain(seats: b);
bb.bookindvidual();
}
```

train

```dart
import 'package:untitledtr/seat.dart';
//parentclass
class Train{
 String ?id;
 List<Seat>seats;
 Train({this.id, required this.seats});



 void bookindvidual(){
   print('booked');
 }
void bookdouble(){
   print('double booked');
}
```

Retrain

```dart
import 'package:untitledtr/seat.dart';
import 'package:untitledtr/train.dart';

class ReTrain extends Train{
  List <String> services=List.empty();
//constructor
  ReTrain({required List<Seat> seats}) : super(seats: seats);

//polymorrp>overide method

  @override
  void bookindvidual(){
    print('booked from retrain');
  }


}
```

yo train

```dart
import 'package:untitledtr/seat.dart';
import 'package:untitledtr/train.dart';

class YoTrain extends Train{
  YoTrain({required List<Seat> seats}) : super(seats: seats);

}
```

seat

```dart
class Seat{
  String ?type;
  String ?price;
  Seat({this.type,this.price});


}
```

……session wedensday

Abstract class train

>not implement method should implement in his son

>not create object of train it for inheritance only

Train

```dart
import 'package:untitledtr/seat.dart';
//parentclass super
abstract class Train{
 String ?id;
```

```
 List<Seat>seats;
 Train({this.id, required this.seats});

//with abstract define method without return
void airconditioner();


 void bookindvidual(){
   print('booked');
 }
void bookdouble(){
   print('double booked');
}
```

retrain

```
class ReTrain extends Train{
  List <String> services=List.empty();
//constructor
  ReTrain({required List<Seat> seats}) : super(seats: seats);

//polymorrp>overide method

  @override
  void bookindvidual(){
    print('booked from retrain');
  }


//abstract for inhertance object define this method
  @override
  void airconditioner(){
    print('air is done');
  }

}
```

yotrain

```
import 'package:untitledtr/seat.dart';
import 'package:untitledtr/train.dart';

class YoTrain extends Train{
  YoTrain({required List<Seat> seats}) : super(seats: seats);
  //abstract for inhertance object define this method that not return in
super class
  @override
  void airconditioner(){
    print('air is done');
  }


}
```

interface

all things in parent class should be override

```
abstract class Seat{
  String ?type;
  String ?price;
  Seat({this.type,this.price});
//if function body should overide also
void getseat(){}


  void pseat();
}

//interface all in super class should overide in son
class TrSeat  implements Seat{
  @override
  String? price;

  @override
  String? type;

  @override
  void getseat() {
    // TODO: implement getseat
  }

  @override
  void pseat() {
    // TODO: implement pseat
  }



}
```

mixcin

mixi. dart

```
mixin behindwindow{
  behind()=>print("seat with behind window");
}

mixin nonwindow{
  behind()=>print("seat with non window");
}
```

seat

```
import 'mixxin.dart';

abstract class Seat{
```

```
  String ?type;
  String ?price;
  Seat({this.type,this.price});
//if function body should overide also
void getseat(){}


  void pseat();
}

//interface all in super class should overide in son
class TrSeat with behindwindow implements Seat {
  @override
  String? price;

  @override
  String? type;
  TrSeat({this.type,this.price}) ;
  @override
  void getseat() {
    // TODO: implement getseat
  }

  @override
  void pseat() {
    // TODO: implement pseat
  }

@override
  behind() {
    // TODO: implement behind
    return super.behind();
  }

}


//interface all in super class should overide in son
class TrSeat2 with nonwindow implements Seat {
  @override
  String? price;

  @override
  String? type;

  @override
  void getseat() {
    // TODO: implement getseat
  }

  @override
  void pseat() {
    // TODO: implement pseat
  }


@override
```

```
  behind() {
    // TODO: implement behind
    return super.behind();
  }
}
```