Circus Of Plates

Islam Mohamed::12

Andrew Adel:: 17

Bahaa khalf::21

Kamal Rashid::

Description:

It is single player-game in which each clown carries two stacks of plates, and there are a set of colored plates queues that fall and he tries to catch them.

How to play:

If you manages to collect three consecutive plates of the same

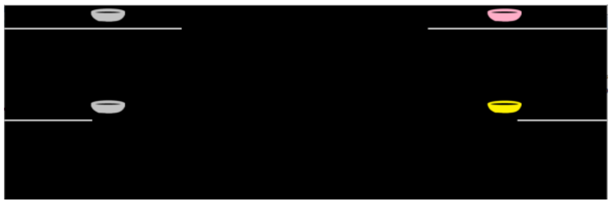
Color or shape , then they are vanished and your score increases.

And the game ends when you collect seven consecutive plates of the same Color or shape.

Design Description:

The game starts at level one.

There is four shelves and the plates start to down from these shelves.



When you collect a plate or it reaches the end of the screen a new plate starts to down from the same shelf .

MenuBar: It contains three menu (File ,Edit,Levels)

File: contain three items.

New game → To start a new game.

Pause → To Pause the game.

Resume → To resume the game.

Edit: contain two items.

Remove → to remove the last shape collected into the stacks.

Add Shape → To dynamically load a Shape and it you can not dynamically load two shape in the same time.

Levels: contain three items.

Level one → To start the game on level one.

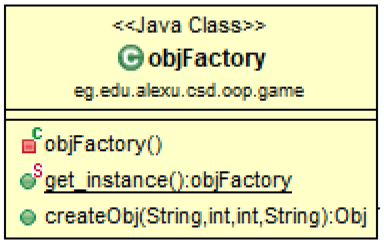
Level two → To start the game on level two.

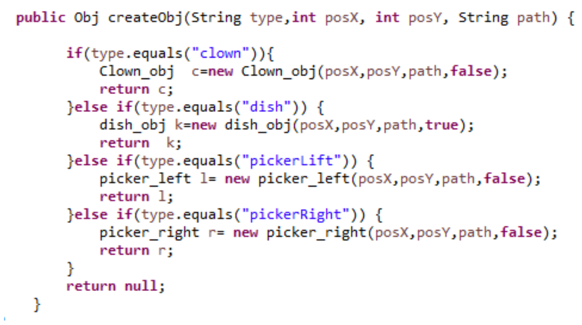
Level three → To start the game on level three.

Design Patterns

Factory:

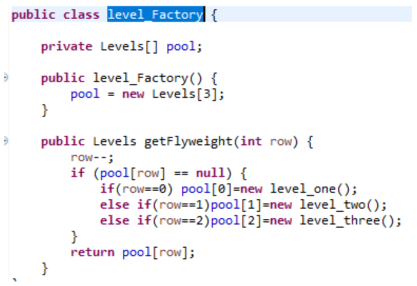
ObjFactory: used to generate GameObjects (clown , dishes,PickerLeft,PickerRight) and we used singleton in it


<<Java Class>>
objFactory
eg.edu.alexu.csd.oop.game
objFactory()
get_instance():objFactory
createObj(String,int,int,String):Obj



```java
public Obj createObj(String type,int posX, int posY, String path) {

      if(type.equals("clown")){
          Clown_obj  c=new Clown_obj(posX,posY,path,false);
          return c;
      }else if(type.equals("dish")) {
          dish_obj k=new dish_obj(posX,posY,path,true);
          return  k;
      }else if(type.equals("pickerLift")) {
          picker_left l= new picker_left(posX,posY,path,false);
          return l;
      }else if(type.equals("pickerRight")) {
          picker_right r= new picker_right(posX,posY,path,false);
          return r;
      }
      return null;
  }
```

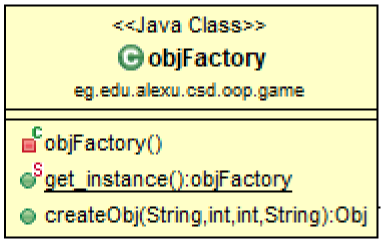
level_Factory: used to generate the levels and we used the Flyweight design pattern in it.



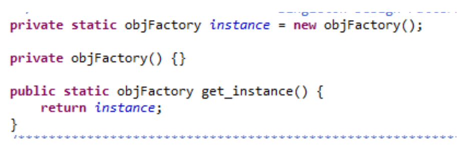
```java
public class level_Factory {

    private Levels[] pool;

    public level_Factory() {
        pool = new Levels[3];
    }

    public Levels getFlyweight(int row) {
        row--;
        if (pool[row] == null) {
            if(row==0) pool[0]=new level_one();
            else if(row==1)pool[1]=new level_two();
            else if(row==2)pool[2]=new level_three();
        }
        return pool[row];
    }
```

Singleton:

we used the singleton in the ObjFactory . to prevent making new instance from this Factory.



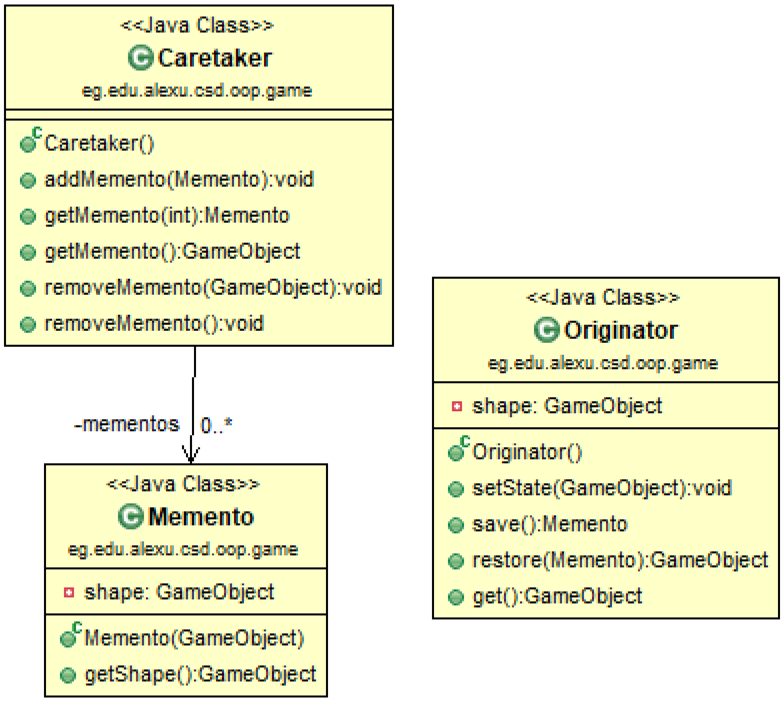
<<Java Class>>
objFactory
eg.edu.alexu.csd.oop.game
objFactory()
get_instance():objFactory
createObj(String,int,int,String):Obj



```
private static objFactory instance = new objFactory();

private objFactory() {}

public static objFactory get_instance() {
    return instance;
}
```

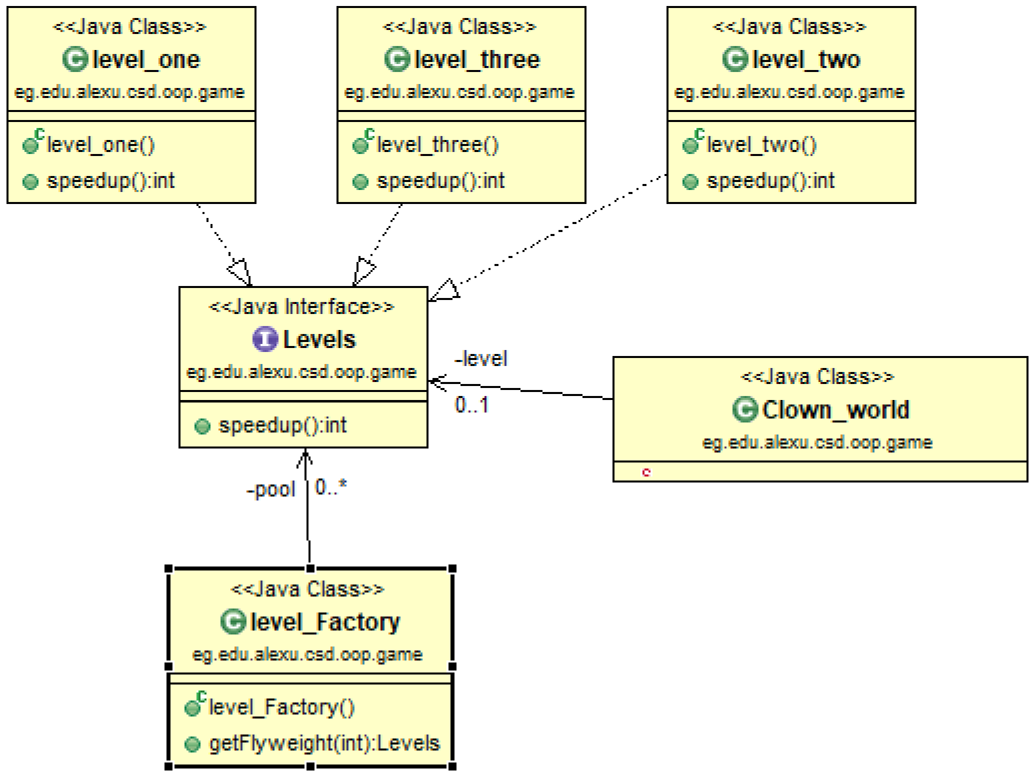
Snapshot:

We use the Snapshot design pattern to the collected dishes in a ArrayList to mange to remove the last one collected.



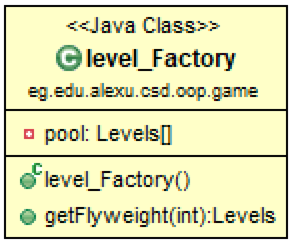
State:

We use state design pattern to make the game levels every level have a method to return the game speed and it is called in the getSpeed() method in the world.



Flyweight:

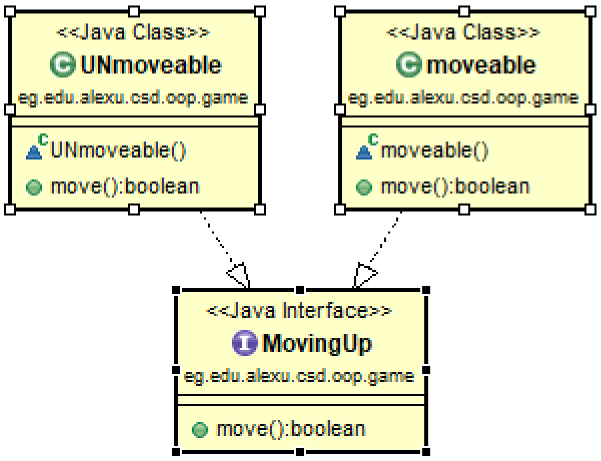
We use the Flyweight design pattern when we make the levels objects in the level_factory to not make many objects from every level.



```
private Levels[] pool;

public level_Factory() {
    pool = new Levels[3];
}

public Levels getFlyweight(int row) {
    row--;
    if (pool[row] == null) {
        if(row==0) pool[0]=new level_one();
        else if(row==1)pool[1]=new level_two();
        else if(row==2)pool[2]=new level_three();
    }
    return pool[row];
}
```

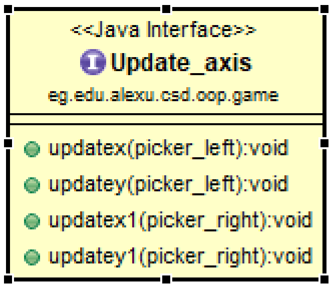
Strategy:

When the clown collect a dish we remove this shape from the moving List and add it to control list to move with the clown but when we do that the dishes become able to move up and down with the arrows and here we use the strategy design pattern to prevent the clown to move up and down and to prevent the dishes to move up and down when it collected by the clown by calling a method that return Boolean if the object can move down.


<<Java Class>>
UNmoveable
eg.edu.alexu.csd.oop.game
UNmoveable()
move():boolean
<<Java Class>>
moveable
eg.edu.alexu.csd.oop.game
moveable()
move():boolean
<<Java Interface>>
MovingUp
eg.edu.alexu.csd.oop.game
move():boolean

Observer:

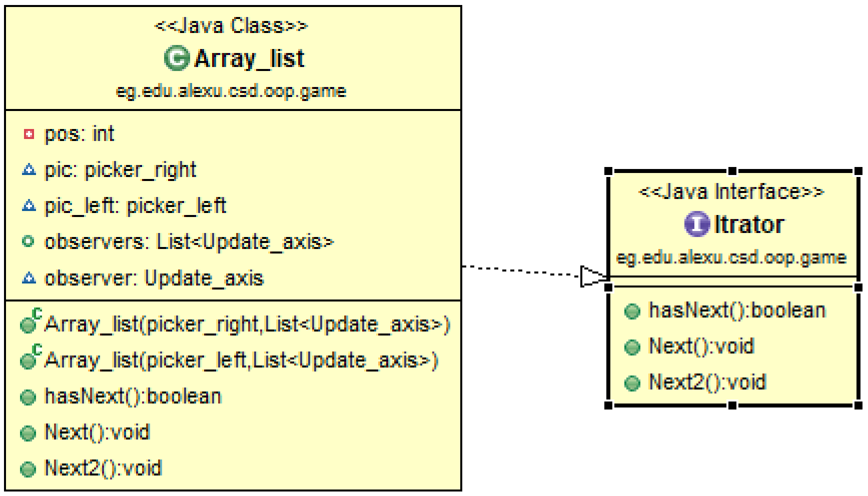
We use the Observer design pattern to know the picker_left && picker_Right X place and send it to all fallen Shapes.



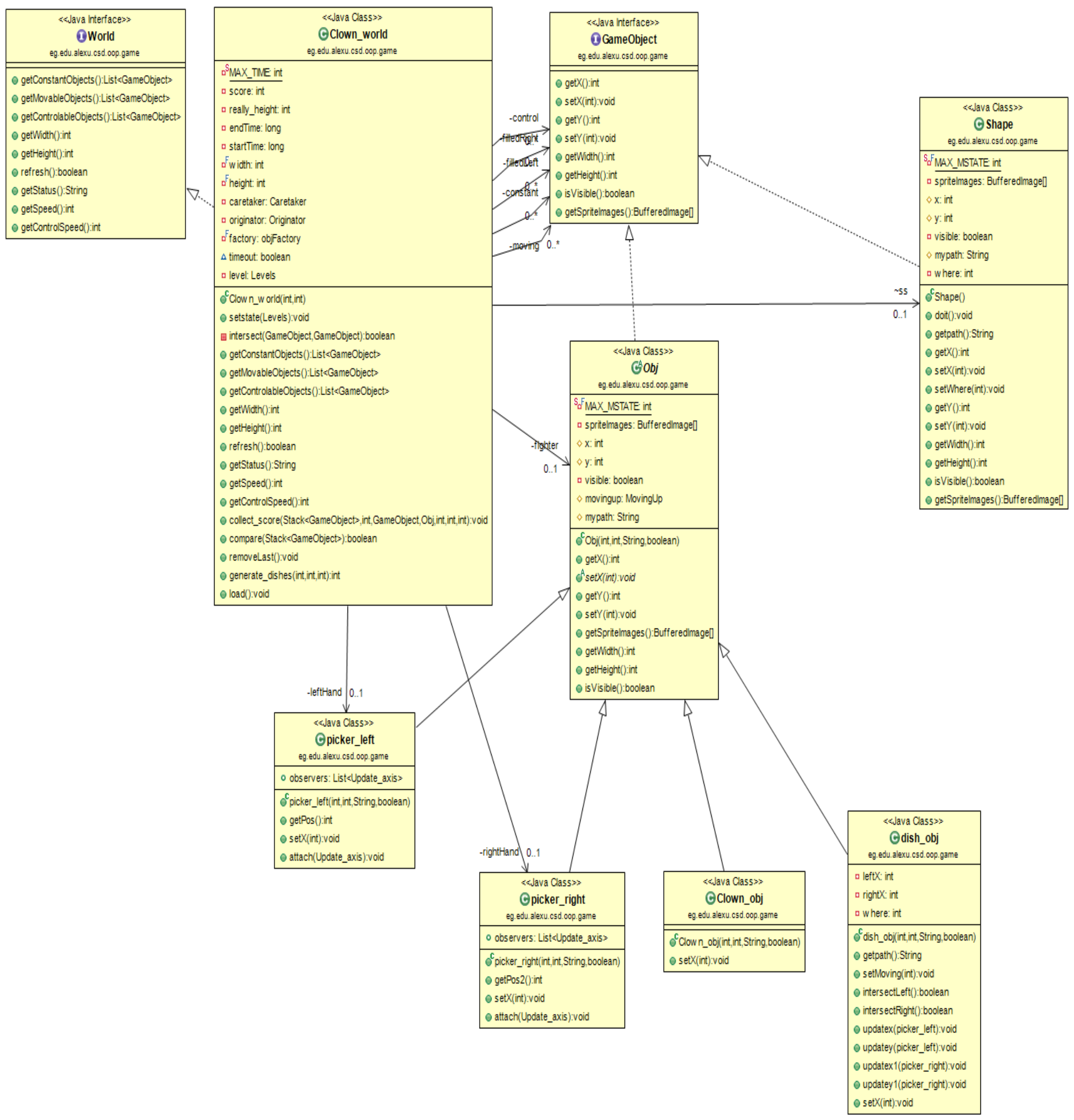
<<Java Interface>>
Update_axis
eg.edu.alexu.csd.oop.game
updatex(picker_left):void
updatey(picker_left):void
updatex1(picker_right):void
updatey1(picker_right):void

Iterator:

We use the iterator design pattern in the picker_left && picker_right to traverse All Observers and Update x_axis and of the pickers .

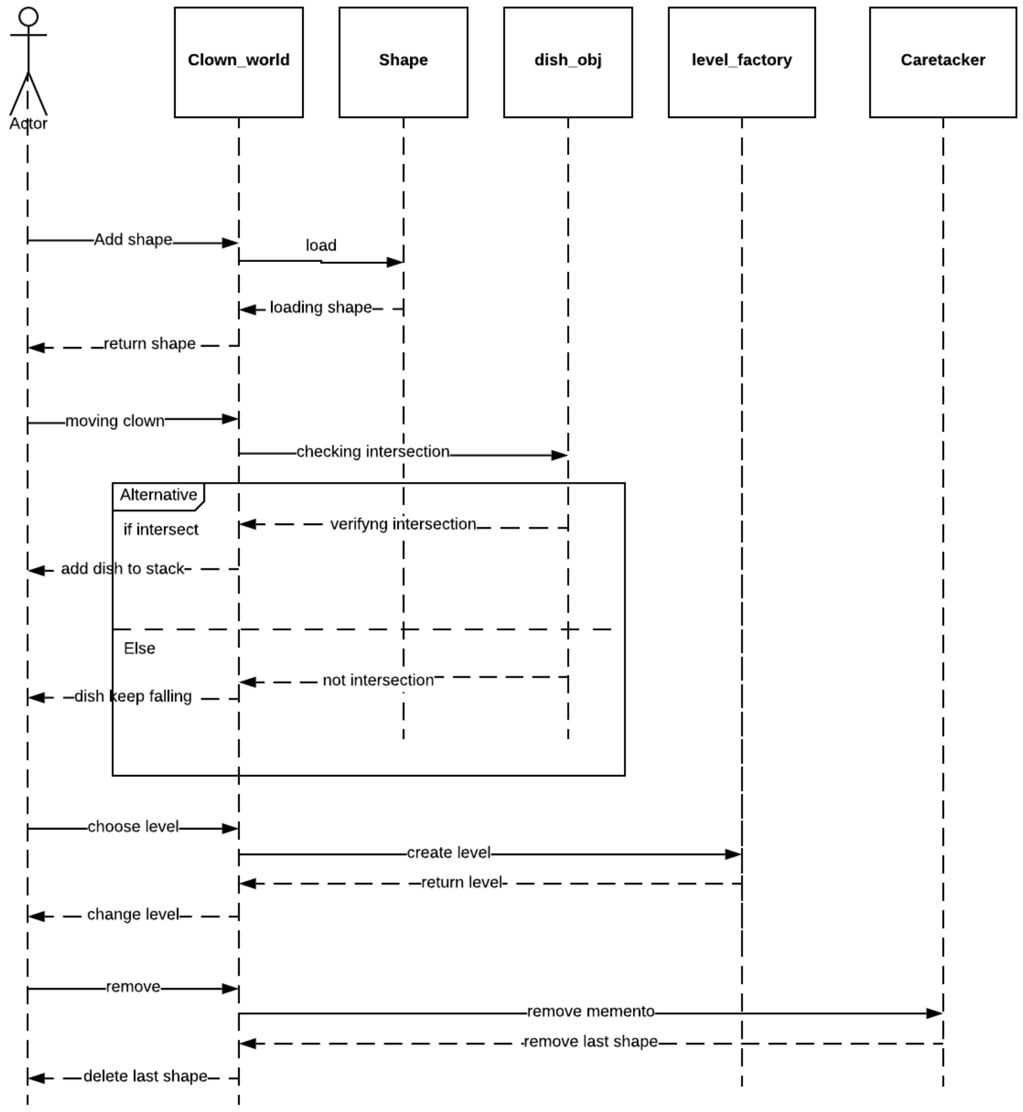


UML



<<Java Interface>>
World
eg.edu.alexu.csd.oop.game
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
refresh():boolean
getStatus():String
getSpeed():int
getControlSpeed():int
<<Java Class>>
Clown_world
eg.edu.alexu.csd.oop.game
MAX_TIME: int
score: int
really_height: int
endTime: long
startTime: long
width: int
height: int
caretaker: Caretaker
originator: Originator
factory: objFactory
timeout: boolean
level: Levels
Clown_world(int,int)
setstate(Levels):void
intersect(GameObject,GameObject):boolean
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
refresh():boolean
getStatus():String
getSpeed():int
getControlSpeed():int
collect_score(Stack<GameObject>,int,GameObject,Obj,int,int,int):void
compare(Stack<GameObject>):boolean
removeLast():void
generate_dishes(int,int,int):int
load():void
<<Java Interface>>
GameObject
eg.edu.alexu.csd.oop.game
getX():int
setX(int):void
getY():int
setY(int):void
getWidth():int
getHeight():int
isVisible():boolean
getSpriteImages():BufferedImage[]
-control
-filledRight
-filledLeft
-constant
0..*
-moving
0..*
<<Java Class>>
Shape
eg.edu.alexu.csd.oop.game
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
mypath: String
where: int
Shape()
doit():void
getpath():String
getX():int
setX(int):void
setWhere(int):void
getY():int
setY(int):void
getWidth():int
getHeight():int
isVisible():boolean
getSpriteImages():BufferedImage[]
~ss
0..1
<<Java Class>>
Obj
eg.edu.alexu.csd.oop.game
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
movingup: MovingUp
mypath: String
Obj(int,int,String,boolean)
getX():int
setX(int):void
getY():int
setY(int):void
getSpriteImages():BufferedImage[]
getWidth():int
getHeight():int
isVisible():boolean
-fighter
0..1
-leftHand
0..1
<<Java Class>>
picker_left
eg.edu.alexu.csd.oop.game
observers: List<Update_axis>
picker_left(int,int,String,boolean)
getPos():int
setX(int):void
attach(Update_axis):void
-rightHand
0..1
<<Java Class>>
picker_right
eg.edu.alexu.csd.oop.game
observers: List<Update_axis>
picker_right(int,int,String,boolean)
getPos2():int
setX(int):void
attach(Update_axis):void
<<Java Class>>
Clown_obj
eg.edu.alexu.csd.oop.game
Clown_obj(int,int,String,boolean)
setX(int):void
<<Java Class>>
dish_obj
eg.edu.alexu.csd.oop.game
leftX: int
rightX: int
where: int
dish_obj(int,int,String,boolean)
getpath():String
setMoving(int):void
intersectLeft():boolean
intersectRight():boolean
updatex(picker_left):void
updatey(picker_left):void
updatex1(picker_right):void
updatey1(picker_right):void
setX(int):void

Sequence Diagram


Actor
Clown_world
Shape
dish_obj
level_factory
Caretacker
Add shape
load
loading shape
return shape
moving clown
checking intersection
Alternative
if intersect
verifyng intersection
add dish to stack
Else
not intersection
dish keep falling
choose level
create level
return level
change level
remove
remove memento
remove last shape
delete last shape