الغاية من الجلسة:

توصيف مسألة البائع الجوال.

مقدمة:

عند البدء ببرمجة أي مسألة هناك خطوات ثابتة نقوم بها :من أجل الترتيب ننشأ حزمة package جديدة ونضع في داخلها صفوف المسألة في هذا المثال سنضع الحزمة في ملف اسمه examples وسيكون اسم الحزمة examples.traveller ثم سننشأ صف جديد اسمه GraphState لنعبر به عن حالات هذه المسألة وبما أنه يعبر عن الحالات فيجب أن يحقق الواجهة State ولكي نحقق الواجهة Stateيجب أن نستوردها import من core.State فإذا أول خطوة عند إنشاء صف الحالات لمسألة هو استيراد core.State ثم جعل الصف يحقق الواجهةState

```
package examples.traveller;
import core.state;
public class GraphState implements State{}
```

وبما أننا نحقق واجهة فيجب أن نعيد تعريف التوابع في الواجهة:

```
package examples.traveller;
import core.state;
public class GraphState implements State{
@Override
    public boolean isGoal(){}
@Override
    public boolean equals(Object s){}
@Override
    public String toString(){}
}
```

هذه الخطوات هي ثابتة لأي مسألة ,الآن كما قلنا سابقا للتعبير عن حالات البيان نحتاج لمصفوفة ثنائية من الأعداد تمثل البيان puplic int[][] theGraph وعدد نخزن فيه الحالة الحالية graphCurrentStateIndexوعدد نحدد من خلاله حجم المصفوفة كي نعرف كمية العناصر التي يجب حجزها graphSize فيصبح الكود:

```
package examples.traveller;
import core.state;
public class GraphState implements State{
    public int graphSize;
    public int graphCurrentStateIndex;
    public int[][] theGraph;
    public GraphState(){}
```

```
        public GraphState(int[][] graph, int currentIndex, int
        size){
                graphCurrentStateIndex=currentIndex;
                graphSize=size;
                theGraph=graph;
}
@Override
        public boolean isGoal(){
                return graphCurrentStateIndex == 3;}
        @Override
        public boolean equals(Object s){
                GraphState temp = (GraphState) s;
                  if(graphCurrentStateIndex !=temp.
        graphCurrentStateIndex)
                        return false;
}
                return true;
}
@Override
public String toString(){
      return "State:\n" +(this.graphCurrentStateIndex + 1);
                                                              }
```

قمنا بتعريف بانيين باني بدون وسطاء لا يقوم بشيء {}graphState() و باني graphState(int[][]graph, int currentIndex, int size)نمرر له مصفوفة ثنائية وعددين هما الحجم والحالة الحالية فيهيئ قيم الصف بها فينشأ حالة من حالات البيان.

ثم قمنا بإعادة تعريف توابع ال واجهة فجعلنا التابع isGoal والذي يرد قيمة بولیانية يختبر إن كنا في الحالة النهائية(والتي فرضا هي 3 مثلاً.)

ثم في التابع equals قصرنا الغرض s من الصف object إلى الصف GraphState لكي نستطيع المقارنة فإن كان رقم الحالة الحالة graphCurrentStateIndex هو نفسه عند غرضين(أي إن كان للحالتين نفس الرقم)فهذه الحالتين هما متساويتين(نفس الحالة.)

وفي التابع toString جعلناه يطبع State ثم ينتقل لسطر جديد ويطبع رقم الحالة (جمعنا له 1 حيث أننا اعتبرنا أن الحالات بدء ترقيمها من 1 بدلاً من صفر .)الخطوة التالية هي تعريف صف يمثل الأفعال actions وسنسميه MoveToNextCity وبما أنا هذا الصف هو يمثل الأفعال فسنحتاج لتحقيق واجهة الأفعال IAction ولكي نستطيع تحقيقها نحتاج استيراد العناصر التالية

وإعادة تعريف توابع core.IAction core.State java.util.ArrayList java.util.Collection
الواجهة , IAction هذه الخطوات ثابتة لصفوف الأفعال في أي مسألة فيصبح الكود:

```
package examples.traveller;
import core.IAction;
import core.State
import java.util.ArrayList;
import java.util.Collection;
public class MoveToNextCity implements IAction{
@Override
	public Collection<State> apply(State s){}
	@Override
	public String getName(){}
}
```

هو هذا مثالنا وفي الفعل اسم يمثل نص سيطبع getName التابع MoveToNextCity .

بينما التابع apply سيرد مجموعة collection من الحالات States التي تحدث عند تطبيق الفعل على حالة ما(s والتي يجب قصرها لتصبح حالة مناسبة لمسألة البائع أي يجب جعلها) graph state أي في مسألة المسافر سيطبق هذا التابع كل الأفعال الممكنة على عقدة وبالتالي سيرد مجموعة العقد التي ترتبط بها هذه العقدة(مثلا في المثال في هذه الجلسة عند تطبيقه على العقدة 1 سيرد العقدتين 2 و3) لمعرفة العقد المرتبطة يكفي المرور على سطر المصفوفة الممثل للحالة(حيث أننا نعرف حجم المصفوفة عبر المتحول) graphSize ورد جميع العقد التي ليست(0 إن مررنا على السطر الأول(والذي نعرفه من خلال) graphCurrentStateIndex في المصفوفة المذكورة في الجلسة سنرى أن العقدتين 2 و 3 قيمهن ليست) 0 فسنقوم بتخزين الحالات بمصفوفة nextStates والتي سيردها التابع apply

```
package examples.traveller;
import core.IAction;
import core.State;
import java.util.ArrayList;
import java.util.Collection;
public class MoveToNextCity implements IAction{
@Override
public Collection<State> apply(State s){
GraphState currentState= (GraphState) s;
ArrayList<State> nextStates = new Arraylist<>();
for(int i=0; i<currentState.graphSize;i++){
```

```
if(currentState.theGraph[currentState.graphCurrentStateIn
dex][i]!=0) {
graphState newState= newGraphState();
```

(إنشاء حالة جديدة عندما لا تكون قيمة العنصر في السطر الخاص بالحالة الحالية تساوي الصفر)

```
newState.graphCurrentIndex=i;
```

(إعطاء رقم الحالة رقم العنصر في السطر)

```
newState.graphSize= currentState.graphSize;
newState.theGraph=currentState.theGraph;
```

(مصفوفة الحالات و حجمها بالنسبة للعقدة المضافة هو نفس الحجم و المصفوفة بالنسبة للعقدة الأصلية.)

```
nextStates.add(newState);
```

(إضافتها لمصفوفة الحالات الجديدة)

```
}
}
Return nextStates;
}
@Override
public String getName(){
return “MoveToNextCity”;}
}
```

بعد ما برمجنا الحالات والأفعال يبقى برمجة المسألة و التي سنضعها في صف اسمه GraphProblem يرث الصف Problem ونحتاج لاستيراد العناصر التالية core.IAction core.Problem core.State core.models.GUIEgde core.models.GUIGraph core.models.GUINode java.util.List java.util.ArrayList ثم إعادة تعريف التوابع الموروثة من الصف(Problem خطوات ثابتة)فيصبح الكود:

```
package example.traveller;
import core.IAction;
import core.Problem;
import core.State;
import core.models.GUIEdge;
import core.models.GUIGraph;
import core.models.GUINode;
```

```
import java.util.ArrayList;
import java.util.List;
public class GraphProblem extends Problem {
@Override
public State getInitialState(){}
@Override
public boolean isGoal(State state){}
@Override
public List<IAction> getActions(){}
@Override
Public double getActionCost(State s,IAction a, State ss){}}
```

نحتاج في صف المسألة للحالة البدائية initialState والحالة النهائيةgoalStateكمتحولات للصف.
فسيرد التابع getInitalState الحالة البدائية وسيختر التابع isGoal إن كانت الحالة state هي نفسها الحالة goalState عبر التابع equals أي(state.equals(goalState
التابع : getActions والذي سيرد لائحة List بالأفعال الممكنة وسنقوم بهذا بالشكل التالي سنخلق به مصفوفة من عناصر الواجهة(IAction وبما أن MoveToNextCityيحقق الواجهة فسنستطيع إضافته)ونضيف الأفعال الممكنةلها.
getActionCostو الذي يعيد التكلفة يحتاج لثلاث وسطاء الحالة الحالية s والحالة الفعل a و الحالة التالية ss حيث تمثل s السطر و ss العمود وبالتالي سنفحص المصفوفة ونرد العنصر المناسب(مثلا في المثال في الجلسة إن كانت s=1 و ss=3سنرد العنصر الثالث في السطر الأول و الذي هو)5 طبعا
نحتاج لقصر الحالات لحالات مناسبة للمسألةGraphStates

```
package example.traveller;
import core.IAction;
import core.Problem;
import core.State;
import core.models.GUIEdge;
import core.models.GUIGraph;
import core.models.GUINode;
import java.util.ArrayList;
import java.util.List;
public class GraphProblem extends Problem {
private GraphState initialState;
private GraphState goalState;
public GraphProblem() {
}
```

```
public GraphProblem(int initialStateIndex, int
goalStateIndex, int[][]) graph {
} @Override
public State getInitialState(){
return initalState;}
@Override
public boolean isGoal(State state){
        return state.equals(goalState};
@Override
public List<IAction> getActions(){
        return new ArrayList<IActions>(){
{
add(new MoveToNextCity());
}
};
}
@Override
Public double getActionCost(State s,IAction a, State ss){
GraphState t= (GraphState)s;
GraphState tt = (GraphState)ss;
return t.theGraph[t.graphCurrentStateIndex][
tt.graphCurrentStateIndex];
```

(نرد هنا قيمة العنصر في السطر t و العمود tt كما شرحنا سابقا)

```
}
}
```

كان هناك وظيفة المطلوب فيها تعريف باني بدون وسطاء وباني بوسطاء هناك عدة طرق لحل هذه الوظيفة مثلا ترك الباني بدون وسطاء فارغا(أي بدون تعريف)وبالنسبة للباني بوسطاء المطلوب تمرير رقم عقدة البداية initialStateIndex و رقم عقدة الهدف goalStateIndex ومصفوفة البيانgraph.

```
public GraphProblem(int initialStateIndex, int
goalStateIndex, int[][] graph){
GraphState start=new
GraphState(graph,initialStateIndex,graph.length());
```

قمنا بإنشاء حالة البداية ومررنا لها مصفوفة البيان ورقم حالة البداية وحجم مصفوفة البيان(راجع باني الصف GraphState)

```
GraphState goal=new
GraphState(graph,goalStateIndex,graph.length());
```

قمنا بنفس الشيء لعقدة النهاية.

```
initialState=start;
goalState=goal;
```

هيئنا الحالة البدائية للصف GraphProblem بحالة البداية والنهاية التي عرفناها

```
}
```

بالنسبة للباني بدون وسطاء يمكننا إعطاء المتحولات قيم اعتباطية افتراضية.

```
public GraphProblem(){
GraphState s = new GraphState();
s.graphSize = 4;
s.graphCurrentStateIndex = 0
s.theGraph = new int[][]{{0,1,1,0},
{1,0,1,0},{1,1,0,1},{0,0,1,0}};
initialState = s;
GraphState temp = new GraphState();
temp.theGraph = s.theGraph;
temp.graphSize = s.graphSize;
temp.graphCurrentStateIndex = s.graphSize – 1;
goalState = temp;
}
```

قمنا هنا بإنشاء حالة البداية s ثم مررنا لها رقما وحجما ومصفوفة للبيان وأسندناها الحالة البدائية للمسألة intialState=s ثم عرفنا حالة temp تمثل حالة النهاية وأعطيناها نفس حجم و مصفوفة s و اعتبرناها آخر عنصر في المصفوفة s.graphSize-1 ثم أسنادها للحالة النهائية.