# CU–FCI–SEII–Midterm–2015-1:15 min (10 marks/20 pts + 2 marks/4 pts bonus)

$\cancel{\phantom{0}}$ 12

Name:........................................................ الرباعى ID: ..........................................

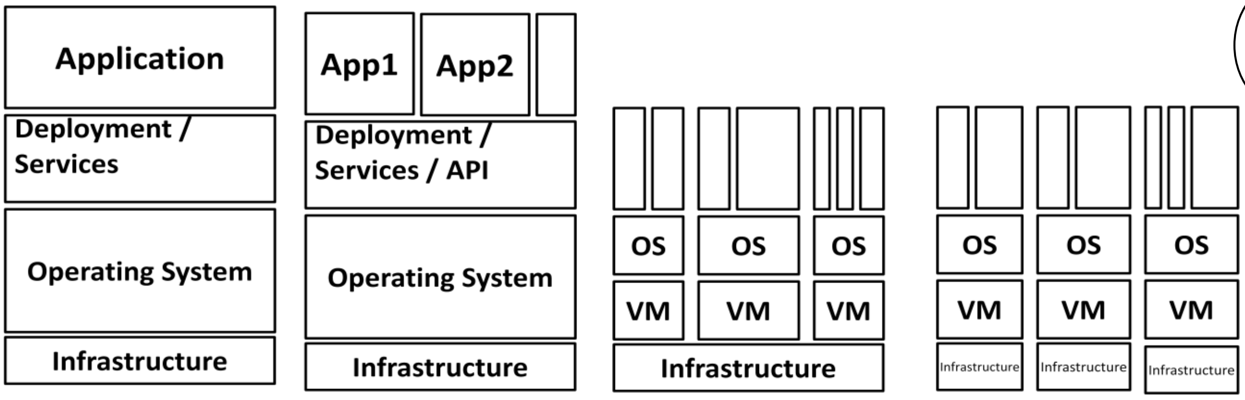
Grp.......................... Email: ........................................................ **Phone**: ........................................

$\cancel{\phantom{0}}$ 24

**Question 1:** Mark by TA: ..................

**Choose the MOST CORRECT and MOST GENEAL answer.** 5 pts

$\cancel{\phantom{0}}$ 5



(a) (b) (c) (d)

1. Which of the diagrams above best describes Google App Engine platform ? (a) [✓ (b)] (c) (d)
2. Which of the diagrams above best describes Software as a Service (SaaS) platforms ? [✓ (a)] (b) (c) (d)
3. Which of the diagrams above best describes Infrastructure as a Service (IaaS) platforms ? (a) (b) [✓ (c)] (d)

Assume that 0 for development/pre-alpha, 1 for alpha, 2 for beta, 3 for release candidate (RC) and 4 for final/production release.

4. Then the first production release of version 1.9 is

   (a) 4.1.9.0 (b) 1.9.1.4 (c) 1.1.9.4 [✓ (d) 1.9.4.0]

5. 1.2.3.4 is

   (a) The first beta release of version 3.4 [✓ (b) The fourth RC release of v 1.2]

   (c) The third production release of v 1.2 (d) The second alpha release of v 3.4

**Question 2:**

$\frac{}{3}$

**Explain three practices** that eXtreme Programming (XP) uses to ensure that the code produced is of good quality. **Explain how each one** of them helps achieve this goal. 3 pts

**Any 3 of these are enough. 1 pt for each, assuming reasonable brief explanation is given. ½ pt for trivial or no explanation.**

- **Collective ownership**: No one person owns or is responsible for individual code segments. In turn, the code is reviewed and updated by everyone on the team, allowing for a more collaborative effort and better code quality.
- **Coding standard**: All team members write code in the same way, using the same styles and formats. This allows for rapid code sharing and improved code quality.
- **Refactoring**: Each member should continually work to adjust and improve the code. This allows the system to be constantly revised without duplicating code.
- **Test Driven Development (Testing first)**: Write the tests first and develop the code to meet the requirements of the test. This allows for a clean application in the long term by flushing out problems before they get lost in a large application.
- **Pair programming**: XP programmers work in pairs. All code is developed by two programmers who work together at a single machine. The expectation is that pair programming produces higher quality code at the same or less cost.
- **Continuous integration**: Software builds are completed several times a day. This keeps all developers on the same page by keeping the application up to date with the most recent coding changes.

**Question 3:**

$\cancel{\phantom{00}}$ **16**

You were hired by Aboudi Soft to design a robot controller system that controls a small robot which navigates through a room. (1) The robot has two types of sensors: ultrasonic sensor and infrared sensor. (2) The ultrasonic sensor uses sound waves to measure distance between the sensor and any objects that exists in front of the sensor. (3) Infrared sensor senses the objects that omit heat like human and animal bodies. (4) The robot is required to navigate through the room without collision تصادم with any object or human/animal beings. (5) Therefore the robot controller has to watch the sensors output. (6) If the robot detects an obstacle, robot controller will perform the following actions to avoid the obstacle: first, it will to turn left and then check (detect) that the way is clear and then move on, if it does not work, it will turn right (from the original position) and then check (detect) that the way is clear and then move on. Finally, if this does not work either (right and left have obstacles), it turn back to its previous position and try again the same steps. (7) The robot will start navigation from a given point **A** and navigate until reaching point **B**. After achieving the point B the robot will do a mission and then it will undo/reverse all its moves to return back to point **A**.

Using what you have learned in design patterns, it is required to design a draft class diagram that could be used as start point for developing the robot controller system.

(1) **Suggest 2 design patterns** that can be used for developing a good design. 2 pt
(2) **Explain in details the role of each pattern** and why it will be used. 2 pts
(3) **Draw the class diagram** for the solution using the design patterns you choose? 8 pts
Show the important methods for each pattern that make the pattern do its task and write a comment on what these methods do or which other methods they call.
(4) **Assume now that we want to change the** design to allow the user to choose from different algorithms to use when collision is detected. For example, instead of left-right-back algorithm described, the user may want to use instead an algorithm that picks a random move and try it every time there is a collision. **Suggest a design pattern** to add this change and explain how to use it **and modify the class diagram** to show how it changes with the use of this pattern. 4 pts

**(1) Observer and Command.** 1 mark for each pattern name. There could be other patterns if an explanation is given in the next part.

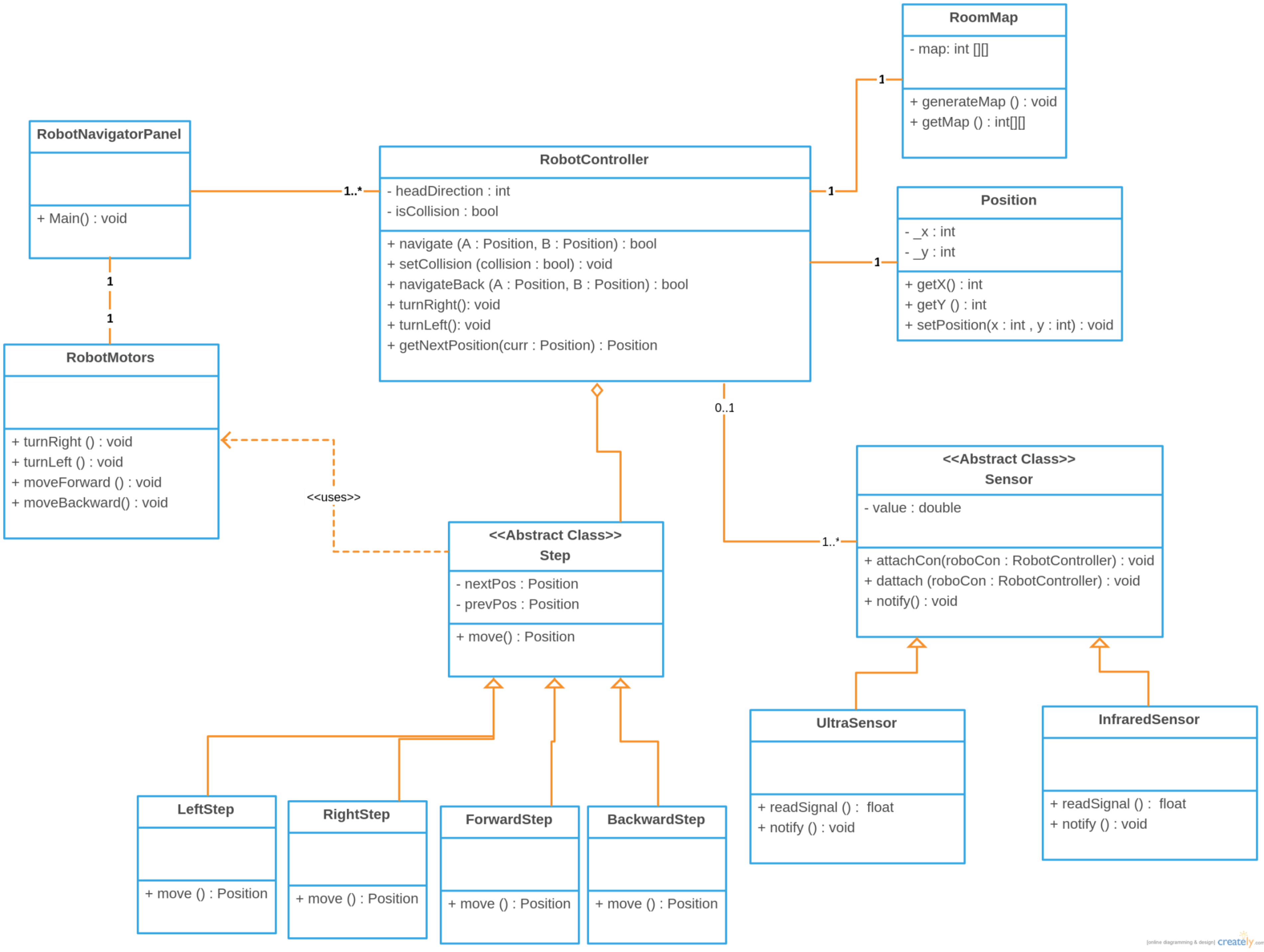
**(2) Role of Observer Pattern.** The controller is the observer and the sensors are the observables. The controller observes their output (their state) for any change that indicate that an obstacle with encountered. (2 marks for any reasonable explanation)

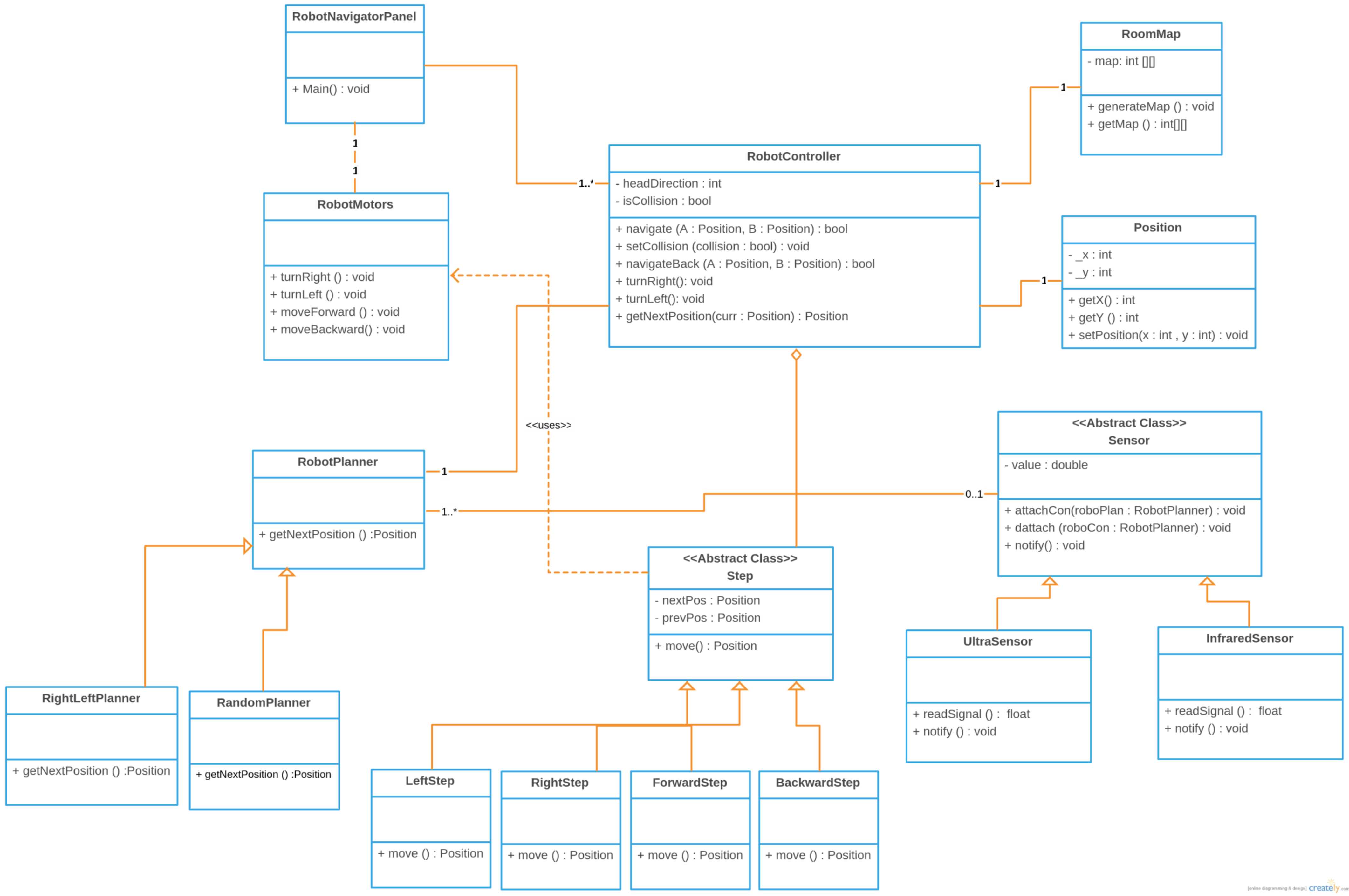
**Role of Command Pattern.** It allows the controller to pass to the invoker commands to execute. (The controller and invoker may be merged in one class). Then the invoker executes the commands and stores them in a stack. Then when the robot reaches its destination, the invoker can pop the commands and undo them. (2 marks for any reasonable explanation)

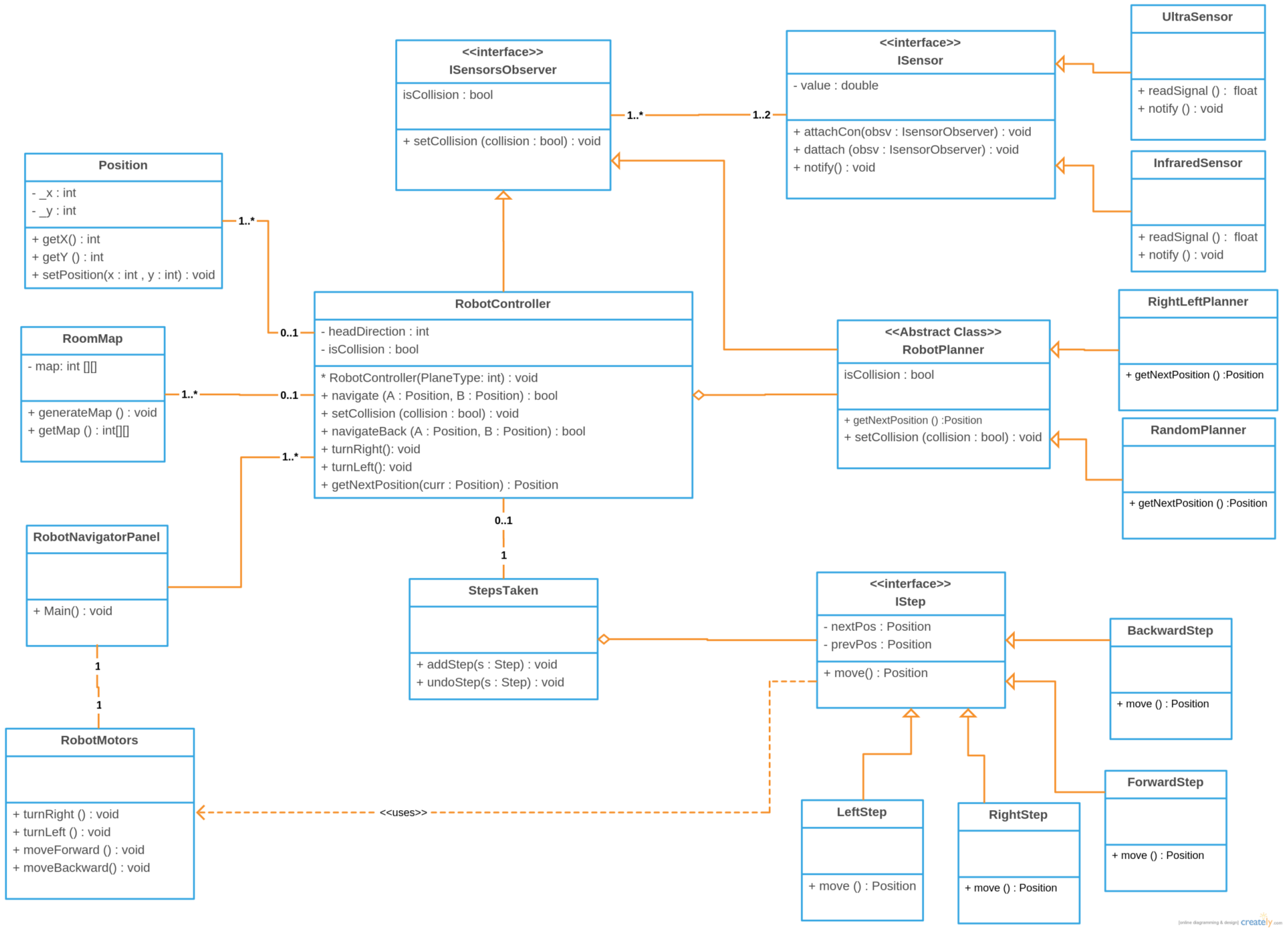
**(4) Role of Strategy Pattern.** The pacontroller is the observer and the sensors are the observables. The controller observes their output (their state) for any change that indicate that an obstacle with encountered. (2 marks for any reasonable explanation)

**(3) and (4) Class Diagram.** Some models are given for guidance but **NOT TO USE FOR EXACT MARKING**. **Extra entity classes like Map, Position, etc** are not important. One model is given with model classes working as design pattern classes as well. Another model is given with pattern abstract classes / interfaces are shown independently.

**For each patter, give 3 marks** for correctly drawing the structure of the pattern (including associations and inheritance relations) and its relation to model (robot system) classes. **Give 1 mark** for correctly showing and overriding the methods (for command: move and undo methods, for observer: attach, detach, notify, for strategy: getNextPosition)


RoomMap
- map: int [][]
+ generateMap () : void
+ getMap () : int[][]
RobotNavigatorPanel
+ Main() : void
RobotController
- headDirection : int
- isCollision : bool
+ navigate (A : Position, B : Position) : bool
+ setCollision (collision : bool) : void
+ navigateBack (A : Position, B : Position) : bool
+ turnRight(): void
+ turnLeft(): void
+ getNextPosition(curr : Position) : Position
Position
- _x : int
- _y : int
+ getX() : int
+ getY () : int
+ setPosition(x : int , y : int) : void
RobotMotors
+ turnRight () : void
+ turnLeft () : void
+ moveForward () : void
+ moveBackward() : void
<<uses>>
<<Abstract Class>>
Step
- nextPos : Position
- prevPos : Position
+ move() : Position
<<Abstract Class>>
Sensor
- value : double
+ attachCon(roboCon : RobotController) : void
+ dattach (roboCon : RobotController) : void
+ notify() : void
1..*
1
1
1
1
1
1
0..1
1..*
LeftStep
+ move () : Position
RightStep
+ move () : Position
ForwardStep
+ move () : Position
BackwardStep
+ move () : Position
UltraSensor
+ readSignal () : float
+ notify () : void
InfraredSensor
+ readSignal () : float
+ notify () : void
[online diagramming & design] creately.com


RobotNavigatorPanel
+ Main() : void
1
1
RobotMotors
+ turnRight () : void
+ turnLeft () : void
+ moveForward () : void
+ moveBackward() : void
1..*
RobotController
- headDirection : int
- isCollision : bool
+ navigate (A : Position, B : Position) : bool
+ setCollision (collision : bool) : void
+ navigateBack (A : Position, B : Position) : bool
+ turnRight(): void
+ turnLeft(): void
+ getNextPosition(curr : Position) : Position
1
1
RoomMap
- map: int [][]
+ generateMap () : void
+ getMap () : int[][]
1
Position
- _x : int
- _y : int
+ getX() : int
+ getY () : int
+ setPosition(x : int , y : int) : void
<<uses>>
1
RobotPlanner
+ getNextPosition () :Position
1..*
0..1
<<Abstract Class>>
Sensor
- value : double
+ attachCon(roboPlan : RobotPlanner) : void
+ dattach (roboCon : RobotPlanner) : void
+ notify() : void
<<Abstract Class>>
Step
- nextPos : Position
- prevPos : Position
+ move() : Position
UltraSensor
+ readSignal () : float
+ notify () : void
InfraredSensor
+ readSignal () : float
+ notify () : void
RightLeftPlanner
+ getNextPosition () :Position
RandomPlanner
+ getNextPosition () :Position
LeftStep
+ move () : Position
RightStep
+ move () : Position
ForwardStep
+ move () : Position
BackwardStep
+ move () : Position



<<interface>>
ISensorsObserver
isCollision : bool
+ setCollision (collision : bool) : void
<<interface>>
ISensor
- value : double
+ attachCon(obsv : IsensorObserver) : void
+ dattach (obsv : IsensorObserver) : void
+ notify() : void
UltraSensor
+ readSignal () : float
+ notify () : void
InfraredSensor
+ readSignal () : float
+ notify () : void
1..*
1..2
Position
- _x : int
- _y : int
+ getX() : int
+ getY () : int
+ setPosition(x : int , y : int) : void
1..*
0..1
RobotController
- headDirection : int
- isCollision : bool
* RobotController(PlaneType: int) : void
+ navigate (A : Position, B : Position) : bool
+ setCollision (collision : bool) : void
+ navigateBack (A : Position, B : Position) : bool
+ turnRight(): void
+ turnLeft(): void
+ getNextPosition(curr : Position) : Position
RoomMap
- map: int [][]
+ generateMap () : void
+ getMap () : int[][]
1..*
0..1
1..*
<<Abstract Class>>
RobotPlanner
isCollision : bool
+ getNextPosition () :Position
+ setCollision (collision : bool) : void
RightLeftPlanner
+ getNextPosition () :Position
RandomPlanner
+ getNextPosition () :Position
RobotNavigatorPanel
+ Main() : void
0..1
1
StepsTaken
+ addStep(s : Step) : void
+ undoStep(s : Step) : void
<<interface>>
IStep
- nextPos : Position
- prevPos : Position
+ move() : Position
BackwardStep
+ move () : Position
1
1
RobotMotors
+ turnRight () : void
+ turnLeft () : void
+ moveForward () : void
+ moveBackward() : void
<<uses>>
LeftStep
+ move () : Position
RightStep
+ move () : Position
ForwardStep
+ move () : Position