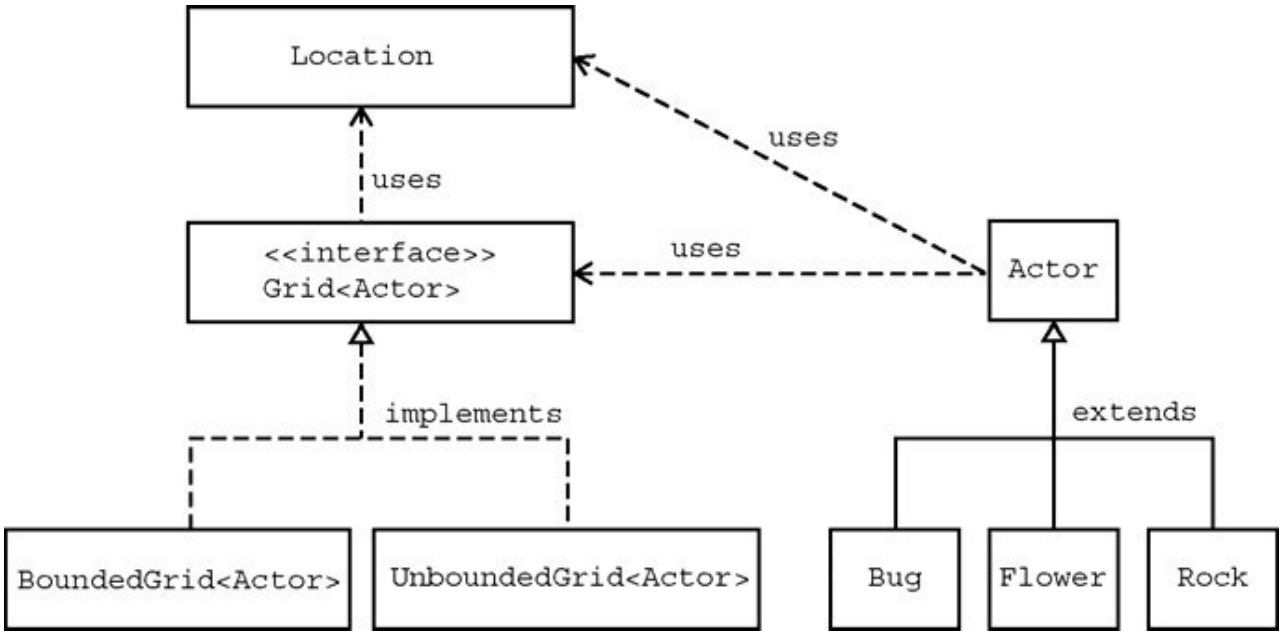
Part 3：GridWorld Classes and Interfaces

被吴沛霖添加，被吴沛霖最后更新于Aug 15, 2015

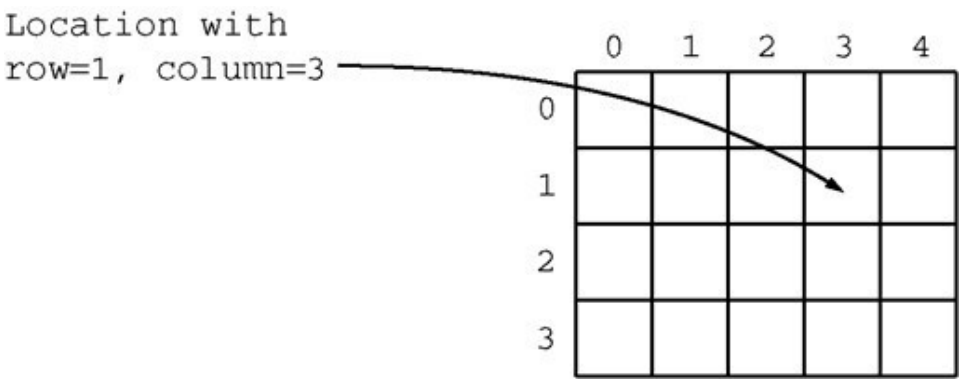
Part 3: GridWorld Classes and Interfaces

In our example programs, a grid contains actors that are instances of classes that extend the *Actor* class. There are two classes that implement the *Grid* interface: *BoundedGrid* and *UnboundedGrid*. Locations in a grid are represented by objects of the *Location* class. An actor knows the grid in which it is located as well as its current location in the grid. The relationships among these classes are shown in the following figure.



The *Location* Class

Every actor that appears has a location in the grid. The *Location* class encapsulates the coordinates for an actor's position in a grid. It also provides methods that determine relationships between locations and compass directions.



Every actor in the grid also has a direction. Directions are represented by compass directions measured in degrees: 0 degrees is north, 45 degrees is northeast, 90 degrees is east, etc. The Location class provides eight constants that specify the compass directions.

```
public static final int NORTH = 0;
public static final int EAST = 90;
public static final int SOUTH = 180;
public static final int WEST = 270;
public static final int NORTHEAST = 45;
public static final int SOUTHEAST = 135;
public static final int SOUTHWEST = 225;
public static final int NORTHWEST = 315;
```

In addition, the *Location* class specifies constants for commonly used turn angles. For example, *Location.HALF_RIGHT* denotes a turn by 45 degrees. Here are the constants for the turn angles.

```
public static final int LEFT = -90;
public static final int RIGHT = 90;
public static final int HALF_LEFT = -45;
public static final int HALF_RIGHT = 45;
public static final int FULL_CIRCLE = 360;
public static final int HALF_CIRCLE = 180;
public static final int AHEAD = 0;
```

To make an actor turn by a given angle, set its direction to the sum of the current direction and the turn angle. For example, the *turn* method of the *Bug* class makes this call.

```
setDirection(getDirection() + Location.HALF_RIGHT);
```

A location in a grid has a row and a column. These values are parameters of the *Location* constructor.

```
public Location(int r, int c)
```

Two accessor methods are provided to return the row and column for a location.

```
public int getRow()
public int getCol()
```

Two other *Location* methods give information about the relationships between locations and directions.

public Location getAdjacentLocation(int direction)

returns the adjacent location in the compass direction that is closest to *direction*

public int getDirectionToward(Location target)

returns the closest compass direction from this location toward target

For example, assume you have the statement below.

```
Location loc1 = new Location(5, 7);
```

The following statements will result in the values indicated by the comments.

```
Location loc2 = loc1.getAdjacentLocation(Location.WEST);
// loc2 has row 5, column 6
Location loc3 = loc1.getAdjacentLocation(Location.NORTHEAST);
// loc3 has row 4, column 8
int dir = loc1.getDirectionToward(new Location(6, 8));
// dir has value 135 (degrees)
```

Note that the row values increase as you go south (down the screen) and the column values increase as you go east (to the right on the screen).

The *Location* class defines the equals method so that *loc.equals*(other) returns *true* if other is a *Location* object with the same row and column values as *loc* and returns *false* otherwise.

The *Location* class implements the *Comparable* interface. The *compareTo* method compares two *Location* objects. The method call *loc.compareTo*(other) returns a negative integer if *loc* has a smaller row coordinate than *other*, or if they have the same row and *loc* has a smaller column coordinate than *other*. The call returns 0 if *loc* and *other* have the same row and column values. Otherwise, the call returns a positive integer.

Do You Know?

Set 3

Assume the following statements when answering the following questions.

```
Location loc1 = new Location(4, 3);
Location loc2 = new Location(3, 4);
```

1. How would you access the row value for loc1?
2. What is the value of b after the following statement is executed?

```
boolean b = loc1.equals(loc2);
```

3. What is the value of loc3 after the following statement is executed?

```
Location loc3 = loc2.getAdjacentLocation(Location.SOUTH);
```

4. What is the value of dir after the following statement is executed?

```
int dir = loc1.getDirectionToward(new Location(6, 5));
```

5. How does the getAdjacentLocation method know which adjacent location to return?

The Grid Interface

The interface Grid<E> specifies the methods for any grid that contains objects of the type E. Two classes, BoundedGrid<E> and UnboundedGrid<E> implement the interface.

You can check whether a given location is within a grid with this method.

boolean isValid(Location loc)

returns true if loc is valid in this grid, false otherwise

Precondition: loc is not null

All methods in this case study have the implied precondition that their parameters are not null. The precondition is emphasized in this method to eliminate any doubt whether null is a valid or invalid location. The null reference does not refer to any location, and you must not pass null to the *isValid* method.

The following three methods allow us to put objects into a grid, remove objects from a grid, and get a reference to an object in a grid.

E put(Location loc, E obj)

puts obj at location loc in this grid and returns the object previously at that location (or null if the location was previously unoccupied)
Precondition: (1) loc is valid in this grid (2) obj is not null

E remove(Location loc)

removes the object at location loc and returns it (or null if the location is unoccupied)
Precondition: loc is valid in this grid

E get(Location loc)

returns the object at location loc (or null if the location is unoccupied)
Precondition: loc is valid in this grid

An additional method returns all occupied locations in a grid.

```
ArrayList<Location> getOccupiedLocations()
```

Four methods are used to collect adjacent locations or neighbor elements of a given location in a grid. The use of these methods is demonstrated by the examples in Part 4.

ArrayList<Location> getValidAdjacentLocations(Location loc)
returns all valid locations adjacent to loc in this grid
Precondition: loc is valid in this grid
ArrayList<Location> getEmptyAdjacentLocations(Location loc)
returns all valid empty locations adjacent to loc in this grid
Precondition: loc is valid in this grid
ArrayList<Location> getOccupiedAdjacentLocations(Location loc)
returns all valid occupied locations adjacent to loc in this grid
Precondition: loc is valid in this grid
ArrayList<E> getNeighbors(Location loc)
returns all objects in the occupied locations adjacent to loc in this grid
Precondition: loc is valid in this grid

Finally, you can get the number of rows and columns of a grid.

```
int getNumRows();
int getNumCols();
```

For unbounded grids, these methods return -1.

Do You Know?

Set 4

1. How can you obtain a count of the objects in a grid? How can you obtain a count of the empty locations in a bounded grid?
2. How can you check if location (10,10) is in a grid?
3. Grid contains method declarations, but no code is supplied in the methods. Why? Where can you find the implementations of these methods?
4. All methods that return multiple objects return them in an ArrayList. Do you think it would be a better design to return the objects in an array? Explain your answer.

The *Actor* Class

The following accessor methods of the Actor class provide information about the state of the actor.

```
public Color getColor()
public int getDirection()
public Grid<Actor> getGrid()
public Location getLocation()
```

One method enables an actor to add itself to a grid; another enables the actor to remove itself from the grid.

```
public void putSelfInGrid(Grid<Actor> gr, Location loc)
public void removeSelfFromGrid()
```

The putSelfInGrid method establishes the actor's location as well as the grid in which it is placed. The removeSelfFromGrid method removes the actor from its grid and makes the actor's grid and location both null.

When adding or removing actors, do not use the put and remove methods of the Grid interface. Those methods do not update the location and grid instance variables of the actor. That is a problem since most actors behave incorrectly if they do not know their location. To ensure correct actor behavior, always use the putSelfInGrid and removeSelfFromGrid methods of the Actor class.

To move an actor to a different location, use the following method.

```
public void moveTo(Location loc)
```

The moveTo method allows the actor to move to any valid location. If the actor calls moveTo for a location that contains another actor, the other one removes itself from the grid and this actor moves into that location.

You can change the direction or color of an actor with the methods below.

```
public void setColor(Color newColor)
public void setDirection(int newDirection)
```

These Actor methods provide the tools to implement behavior for an actor. Any class that extends Actor defines its behavior by overriding the act method.

```
public void act()
```

The act method of the Actor class reverses the direction of the actor. You override this method in subclasses of Actor to define actors with different behavior. If you extend Actor without specifying an act method in the subclass, or if you add an Actor object to the grid, you can observe that the actor flips back and forth with every step.

The Bug, Flower, and Rock classes provide examples of overriding the act method.

The following questions help analyze the code for Actor.

Do You Know?

Set 5

1. Name three properties of every actor.
2. When an actor is constructed, what is its direction and color?
3. Why do you think that the Actor class was created as a class instead of an interface?
4. Can an actor put itself into a grid twice without first removing itself? Can an actor remove itself from a grid twice? Can an actor be placed into a grid, remove itself, and then put itself back? Try it out. What happens?
5. How can an actor turn 90 degrees to the right?

Extending the Actor Class

The Bug, Flower, and Rock classes extend Actor in different ways. Their behavior is specified by how they override the act method.

The Rock Class

A rock acts by doing nothing at all. The act method of the Rock class has an empty body.

The Flower Class

A flower acts by darkening its color, without moving. The act method of the Flower class reduces the values of the red, green, and blue components of the color by a constant factor.

The Bug Class

A bug acts by moving forward and leaving behind a flower. A bug cannot move into a location occupied by a rock, but it can move into a location that is occupied by a flower, which is then removed. If a bug cannot move forward because the location in front is occupied by a rock or is out of the grid, then it turns right 45 degrees.

In Part 2, exercises were given to extend the Bug class. All of the exercises required the act method of the Bug class to be overridden to implement the desired behavior. The act method uses the three auxiliary methods in the Bug class: canMove, move, and turn. These auxiliary methods call methods from Actor, the superclass.

The canMove method determines whether it is possible for this Bug to move. It uses a Java operator called instanceof (not part of the AP CS Java subset). This operator is used as in the following way.

expr instanceof Name

Here, expr is an expression whose value is an object and Name is the name of a class or interface type. The instanceof operator returns true if the object has the specified type. If Name is a class name, then the object must be an instance of that class itself or one of its subclasses. If Name is an interface name, then the object must belong to a class that implements the interface.

This statement in the canMove method checks whether the object in the adjacent location is null or if it is a Flower.

```
return (neighbor == null) || (neighbor instanceof Flower);
```

The following statement in the canMove method checks that the bug is actually in a grid---it would be possible for the bug not to be in a grid if some other actor removed it.

```
if (gr == null) return false;
```

```
The other code for canMove is explored in the questions at the end of this section.
```

```
The move method for Bug moves it to the location immediately in front and puts a flower in its
previous location. The turn method for Bug turns it 45 degrees to the right. The code for these
methods is explored in the following questions.
```

Do You Know?
Set 6

1. Which statement(s) in the canMove method ensures that a bug does not try to move out of its grid?
2. Which statement(s) in the canMove method determines that a bug will not walk into a rock?
3. Which methods of the Grid interface are invoked by the canMove method and why?
4. Which method of the Location class is invoked by the canMove method and why?
5. Which methods inherited from the Actor class are invoked in the canMove method?
6. What happens in the move method when the location immediately in front of the bug is out of the grid?
7. Is the variable loc needed in the move method, or could it be avoided by calling getLocation() multiple times?
8. Why do you think the flowers that are dropped by a bug have the same color as the bug?
9. When a bug removes itself from the grid, will it place a flower into its previous location?
10. Which statement(s) in the move method places the flower into the grid at the bug's previous location?
11. If a bug needs to turn 180 degrees, how many times should it call the turn method?

Group Activity

Organize groups of 3--5 students.

1. Specify: Each group creates a class called Jumper. This actor can move forward two cells in each move. It "jumps" over rocks and flowers. It does not leave anything behind it when it jumps.

   In the small groups, discuss and clarify the details of the problem:
   a. What will a jumper do if the location in front of it is empty, but the location two cells in front contains a flower or a rock?
   b. What will a jumper do if the location two cells in front of the jumper is out of the grid?
   c. What will a jumper do if it is facing an edge of the grid?
   d. What will a jumper do if another actor (not a flower or a rock) is in the cell that is two cells in front of the jumper?
   e. What will a jumper do if it encounters another jumper in its path?
   f. Are there any other tests the jumper needs to make?
2. Design: Groups address important design decisions to solve the problem:
   a. Which class should Jumper extend?
   b. Is there an existing class that is similar to the Jumper class?
   c. Should there be a constructor? If yes, what parameters should be specified for the constructor?
   d. Which methods should be overridden?
   e. What methods, if any, should be added?
   f. What is the plan for testing the class?
3. Code: Implement the Jumper and JumperRunner classes.
4. Test: Carry out the test plan to verify that the Jumper class meets the specification.

What Makes It Run? (Optional)

A graphical user interface (GUI) has been provided for running GridWorld programs. The World class makes the connection between the GUI and the classes already described. The GUI asks the world for its grid, locates the grid occupants, and draws them. The GUI allows the user to invoke the step method of the world, either taking one step at a time or running continuously. After each step, the GUI redisplays the grid.

For our actors, a subclass of the World class called ActorWorld is provided. ActorWorld defines a step method that invokes act on each actor.

Other worlds can be defined that contain occupants other than actors. By providing different implementations of the step method and other methods of the World class, one can produce simulations, games, and so on.

In order to display the GUI, a runner program constructs a world, adds occupants to it, and invokes the show method on the world. That method causes the GUI to launch.

The ActorWorld class has a constructor with a Grid<Actor> parameter. Use that constructor to explore worlds with grids other than the default 10 x 10 grid.

The ActorWorld has two methods for adding an actor.

```
public void add(Location loc, Actor occupant)
public void add(Actor occupant)
```

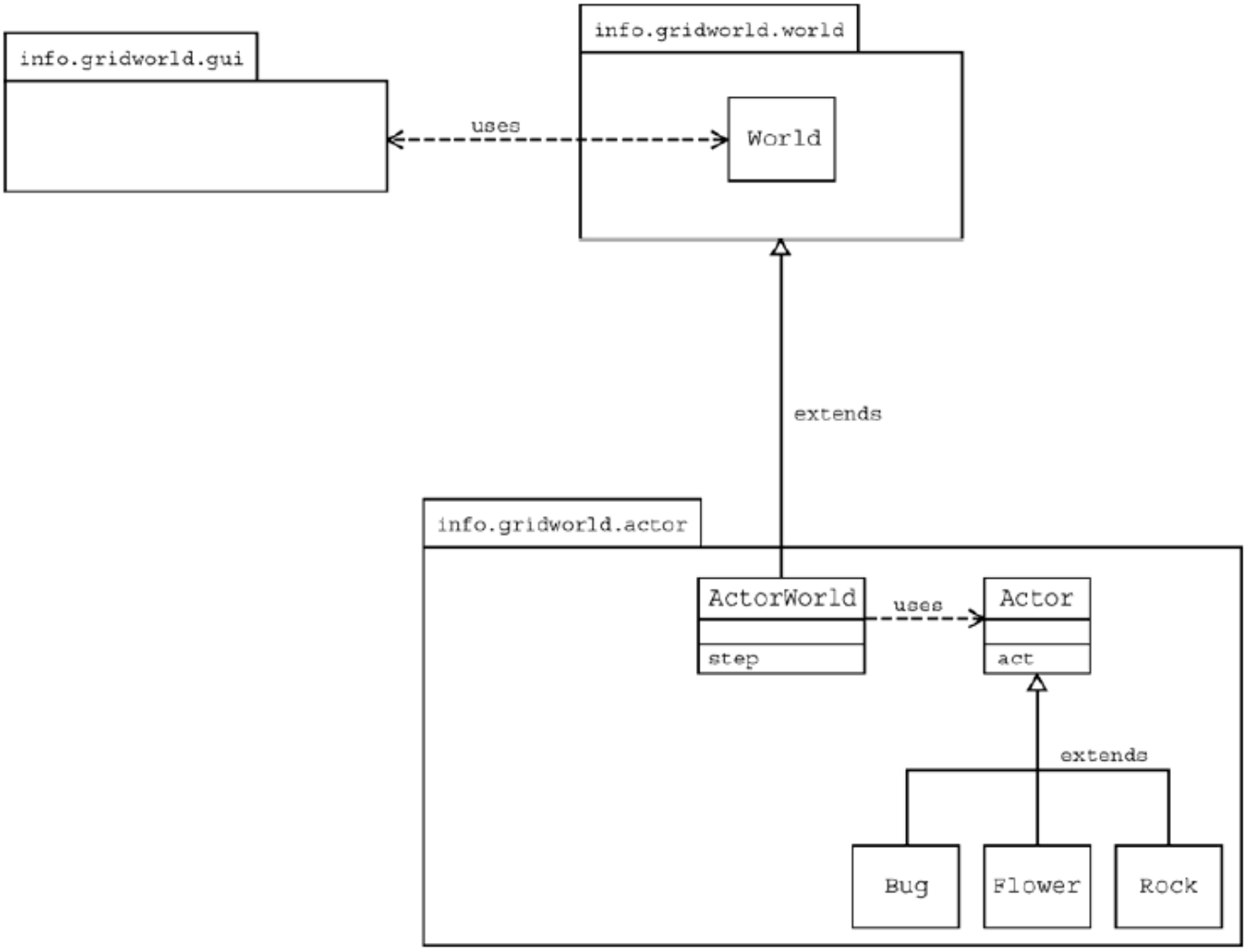
The add method without a Location parameter adds an actor at a random empty location.

When adding actors to a world, be sure to use the add method of the ActorWorld class and not the put method of the Grid interface. The add method calls the Actor method putSelfInGrid. As explained previously, the putSelfInGrid method sets the actor's references to its grid and location and calls the grid method put, giving the grid a reference to the actor.

The remove method removes an actor from a given location and returns the Actor that has been removed.

```
public Actor remove(Location loc)
```

The relationship between the GUI, world, and actor classes is shown in the following figure. Note that the GUI has no knowledge of actors. It can show occupants in any world. Conversely, actors have no knowledge of the GUI.



Like 2 people like this. None

评论

林育新 发表: Aug 20, 2015

It "jumps" over rocks and flowers. It does not leave anything behind it when it jumps.

I think that it acts like shanxian

殷家康 发表: Aug 20, 2015

R闪回旋踢

田满鑫 发表: Aug 20, 2015

如果说看英文要有点技巧， 我会加点旋律 // 你会觉得 超diao~~~~~

郑詠心 发表: Aug 20, 2015

好了，好了，我知道了。

李展文 发表： Aug 20, 2015

haole, haole, wo zhidao le.

郑詠心 发表： Aug 21, 2015

ni ma zha l

莫冠钊 发表： Aug 21, 2015

nima zhale

龙德成 发表： Aug 21, 2015

- **if you can see this ,you will kandao *"ni shi SB"***

何朝东 发表： Aug 20, 2015

O

袁晓晖 发表： Aug 20, 2015

QQ2WRD，一库！

对不起，我是中大第一神瞎。

不要说话，吻我。

张志翔 发表： Aug 21, 2015

sha gou

申明 发表： Aug 21, 2015

+25 gold

刘健坤 发表： Aug 20, 2015

Happy Double Seventh Festival , coders~

罗思成 发表： Aug 20, 2015

第3部分：GridWorld类和接口
在我们的例程中，一个网格包含行为者是类扩展了演员类的实例。有迹象表明，实现了网格接口两个类：BoundedGrid和UnboundedGrid。在网格位置由位置类的对象表示。一个演员知道在它所处以及在网格当前位置网格。这些类之间的关系如图所示。

Location类
出现的每个行为人具有在网格中的位置。该位置类封装了坐标网格中的一个演员的地位。它还规定，确定位置和罗盘方向之间的关系的方法。

网格中的每个演员也有一个方向。方向是由以度罗盘方位代表：0度是北，45度的东北，90度是东，等等。Location类提供了指定罗盘方位8常量。

公共静态最终诠释北= 0;
公共静态最终诠释EAST = 90;
公共静态最终诠释，中南= 180;
公共静态最终诠释WEST = 270;
公共静态最终诠释东北= 45;
公共静态最终诠释东南= 135;
公共静态最终诠释，西南= 225;
公共静态最终诠释，西北= 315;
此外，位置类指定为常用转弯角度常数。例如，Location.HALF_RIGHT表示转45度。下面是转弯角度的常量。

公共静态最终诠释LEFT = -90;
公共静态最终诠释RIGHT = 90;
公共静态最终诠释HALF_LEFT = -45;
公共静态最终诠释HALF_RIGHT = 45;
公共静态最终诠释FULL_CIRCLE = 360;
公共静态最终诠释HALF_CIRCLE = 180;
AHEAD公共静态最终诠释= 0;
为了使一个演员又由一个给定的角度，设置其方向与电流方向的总和和转弯角度。例如，该错误类的转弯方法使此呼叫。

setDirection（getDirection（）+ Location.HALF_RIGHT）；
在网格甲位置上有一排和列。这些值是位置构造函数的参数。

公共场所（INT R，INT C）
设置两个存取方法返回的行和列的位置。

公众诠释中的getRow（）
公众诠释getCol（）
另外两个位置的方法给出关于位置和方向之间的关系的信息。

公共场所getAdjacentLocation（INT方向）

返回相邻的位置，在罗盘方向最接近的方向

公众诠释getDirectionToward（目标定位）

从这个位置朝目标返回最接近的罗盘方向

例如，假设你有下面的语句。

位置LOC1 =新位置（5,7）；
下面的语句将导致注释所指示的数值。

位置LOC2 = loc1.getAdjacentLocation（Location.WEST）；
// LOC2有5行，第6列
定位中Loc3 = loc1.getAdjacentLocation（Location.NORTHEAST）；
//中Loc3有行4列8
INT DIR = loc1.getDirectionToward（新位置（6,8））；
//目录的值为135（度）
需要注意的是该行的值增加，因为你去南方（下屏幕）和列值增加，因为你往东走（在右边屏幕上）。

该位置类定义equals方法，这样，如果对方是用相同的行和列的值作为LOC Location对象否则，返回假loc.equals（其他）返回true。

该位置类实现Comparable接口。 CompareTo方法比较两个位置的对象。方法调用loc.compareTo（其他）返回一个负整数如果禄具有比其他更小的行坐标，或比其他如果它们具有相同的行和禄具有较小的柱坐标。调用返回0，如果禄和其他有相同的行和列的值。否则，调用返回一个正整数。

你知道吗?

集3

回答下列问题时，假设下面的语句。

位置LOC1 =新位置（4，3）；
位置loc2的=新位置（3，4）；
1.你将如何访问行值LOC1?
2.什么是b的值执行下面的语句之后?

布尔B = loc1.equals（LOC2）；
3.什么是中Loc3的值执行下面的语句之后?

定位中Loc3 = loc2.getAdjacentLocation（Location.SOUTH）；
4.什么是DIR值执行下面的语句之后?

INT DIR = loc1.getDirectionToward（新位置（6,5））；
5.如何在getAdjacentLocation方法知道要返回的相邻位置?

网格接口
接口网格<E>指定方法包含E.两类，BoundedGrid <E>和UnboundedGrid <E>实现该接口类型的对象的任何电网。

可以检查给定的位置是否与该方法的网格内。

布尔的isValid（位置LOC）

如果LOC是有效的在这个网格中，否则为false返回true

前提条件：禄不为空

在这种情况下，研究中的所有方法都隐含的前提是其参数不为空。前提是强调了这种方法，以消除任何疑问空是否是有效的还是无效的位置。空引用不引用任何位置，你不能传递null给isValid方法。

以下三种方法可以让我们把对象变成一个网格，从网格中删除的对象，并获得在网格中引用的对象。

e将（位置禄，E OBJ）

把OBJ在位置LOC在此网格并返回对象以前在那个位置（或NULL，如果位置是以前无人居住）
前提条件：（1）loc在此网格有效（2）obj不为null

e卸下（位置LOC）

除去在位置LOC的对象，并返回它（或NULL，如果位置是无人居住）
前提条件：LOC在此网格有效

ËGET（位置LOC）

返回对象在位置LOC（或NULL，如果位置是无人居住）
前提条件：LOC在此网格有效

另外一个方法返回一个网格占据的位置。

的ArrayList <位置> getOccupiedLocations（）
四种方法用于收集相邻位置或在网格的给定位置的相邻元素。使用这些方法是在第4部分表明的实施例。

的ArrayList <位置> getValidAdjacentLocations（位置LOC）
返回紧邻禄所有有效地点在该网格
前提条件：LOC在此网格有效
的ArrayList <位置> getEmptyAdjacentLocations（位置LOC）
返回此网格紧邻禄所有有效的空位置
前提条件：LOC在此网格有效
的ArrayList <位置> getOccupiedAdjacentLocations（位置LOC）
返回紧邻禄所有有效占领的地点在该网格
前提条件：LOC在此网格有效
的ArrayList <E> getNeighbors（位置LOC）
返回所有对象，在此网格毗邻禄被占领的地点
前提条件：LOC在此网格有效
最后，可以得到一个网格的行和列的数目。

诠释getNumRows（）；
诠释getNumCols（）；
对于无界网格，这些方法返回-1。

你知道吗?

设置4

1.如何获得在网格中的对象的数量？如何获得在有界格的空位置计数?
2.如何检查位置（10,10）是在网格?
3.网格包含方法的声明，但没有代码的方式提供。为什么呢？哪里可以找到这些方法的实现?
4.返回多个对象的所有方法都返回他们在一个ArrayList中。你是否认为这将是一个更好的设计来返回对象的数组？解释你的答案。

Actor类
在演员类的以下的存取方法提供有关演员的状态信息。

众彩的getColor（）
公众诠释getDirection（）
公共电网<演员> getGrid（）
公共场所的getLocation（）
一种方法是使演员将自身添加到网格;另一个让演员本身，从网格中删除。

公共无效putSelfInGrid（网格<演员>克，地点LOC）
公共无效removeSelfFromGrid（）
所述putSelfInGrid方法建立演员的位置，以及在其中它被放置在网格。该removeSelfFromGrid方法删除其电网演员和做演员的电网和位置两个空。

当添加或删除角色，不要使用put并删除网格接口的方法。这些方法不更新演员的位置和网格实例变量。这是一个问题，因为多数演员出现错误的行为，如果他们不知道他们的位置。为确保正确演员的行为，总是使用Actor类的putSelfInGrid和removeSelfFromGrid方法。

要移动一个演员到不同的位置，使用下面的方法。

公共无效MOVETO（位置LOC）
moveTo方法可以让演员来移动到任何有效位置。如果演员调用MOVETO对于包含一个行为者的位置，另一种从网格删除自身和此演员移动到该位置。

您可以更改与下面的方法演员的方向或颜色。

公共无效setColor（彩色newColor）
公共无效setDirection（INT newDirection）
这些演员的方法提供工具，以实现对一个演员的行为。扩展演员任何类通过重写行为方法定义它的行为。

公共无效的行为（）
Actor类的行为方法将反转演员的方向。您可以在演员的子类覆盖此方法来定义不同的行为者。如果扩展演员不指定在子类的行为方式，或者如果你是演员对象添加到网格中，你可以观察到演员翻转来回的每一步。

重新设计的Bug，花，和岩石类提供重写行为方法的例子。

下列问题有助于分析对演员的代码。

你知道吗?

设置5

1.名称每个演员的三个属性。
2.当一个演员构造，什么是它的方向和颜色?
3.为什么你认为演员类是创建一个类，而不是一个接口?
4.可以当演员把自己变成一个网格两次没有先删除自己？一个演员可以从电网自身删除两次？可一个演员被放置到一个网格，自身删除，然后把自己回来？试试看。会发生什么?
5.如何能当演员转90度，向右?

扩展演员类
重新设计的Bug，花，和岩石类以不同的方式扩展演员。由它们如何重写行为法规定的行为。

岩石类

一个摇滚的行为做什么都没有。岩石类的行为方法有一个空的机构。

花卉类

花行径变暗的颜色，没有移动。花卉类的动作方法由一个常数因子减少的颜色的红色，绿色和蓝色分量的值。

Bug类

一个错误行径向前迈进，并留下了一朵花。一个错误不能移动到由岩石占据的位置，但它可以移动到由花，然后除去所占据的位置。如果一个错误无法前进，因为在前面的位置由岩石或占用超出电网的，那么它向右转45度。

在第二部分，演习给予延长Bug类。所有的练习需要重写Bug类的行为方法来实现所期望的行为。该法案方法使用中的Bug类中的三个辅助方法：canMove，移动和转动。这些辅助方法调用从演员的方法，超。

所述canMove方法确定是否有可能为这个Bug移动。它使用Java运算符叫的instanceof（不属于AP CS Java的子集）。这个操作符被用作以下列方式。

EXPR的instanceof名称

在这里，expr是它的值是一个对象，名称是一个类或接口类型的名称的表达式。如果对象具有特定类型的instanceof运算符返回true。如果名称是一个类名，那么对象必须是类本身或者它的一个子类的实例。如果名称是一个接口名，那么对象必须属于一个实现该接口的类。

在canMove方法检查该语句是否在相邻位置的对象为null，或者如果它是一朵花。

返程（邻居== NULL）||（邻居的instanceof花）；
在canMove方法下面的语句检查该错误实际上是在网格---这将是可能的错误不会在网格，如果其他演员删除它。

如果（GR == NULL）返回false;
其他代码canMove进行了探索在本节末尾的问题。
此举方法错误，立即在前面将其移动到的位置，并把一朵花插在它以前的位置。转方法错误把它45度的权利。对于这些方法的代码进行了探索在以下问题。
你知道吗?
设置6

1.哪些在canMove方法声明（S），确保了错误不会尝试将其格?
在canMove方法2.哪种说法（S）确定一个bug会不会走进一个石头?
3.其中的格界面的方法是由canMove方法，为什么调用?
位置类4.哪种方法由canMove方法，为什么调用?
5.从演员类继承哪些方法在canMove方法调用?
6.在移动方法会发生什么时，立刻在错误前面的位置是出了格的?
7.是否需要在移动方法变量禄，或可以被避免调用的getLocation（）多次?
8.为什么你认为被错误丢弃的花朵有相同颜色的错误?
9.当一个bug从网格删除自身，将其放置一朵花到其先前的位置?
中招方法10.下列哪项（S）地方花入电网的缺陷先前的位置?
11.如果一个bug需要转180度，多少次它应该叫转法?

小组活动
组织的3-5的学生群体。

1.指定：每一组创建一个类跳线。这演员可以在两种细胞前进中的每一个举动。据“跳跃”在岩石和鲜花。它不会离开它背后的东西，当它跳跃。

在小团体，讨论和澄清这一问题的详细信息：
一个。什么将跳线做，如果在它前面的位置是空的，但位置两个单元格前面包含一花或摇滚？
湾什么将一个跳线做，如果位置的两个单元格中的跳线面前的是出了格的？
C。什么将一个跳线做，如果它是面向网格的边缘？
ð。什么将跳线做，如果另一位演员（而不是花或岩石）是是两个单元的跳线前面的小区？
即什么会如果遇到在道路上又跳投命中一记跳投吗？
F。是否有任何其他的测试跳线需要做什么呢？
2.设计：用户解决重要的设计决策来解决问题：
一个。哪个类应该跳线扩展？
湾是否有一个现有的类，它是类似于跳线类？
C。应该有一个构造函数？如果是，哪些参数应该在构造函数中指定？
ð。哪些方法应该重写？
即什么方法，如果有的话，应该补充？
F。什么是计划测试的类？
3.代码：实现跳线和JumperRunner类。
4.测试：执行测试计划，以验证跳线类符合规范。

是什么使得它运行？ （可选）
的图形用户界面（GUI）已提供用于运行GridWorld程序。世界一流使得已经描述的图形用户界面和类之间的连接。图形用户界面询问世界的网格，定位网格住户，并提请他们。该GUI允许用户调用世界的步骤的方法，或者采取一个步骤在一个时间或连续地运行。每个步骤之后，在GUI重新显示网格。

对于我们的演员，叫ActorWorld世界类的子类提供。 ActorWorld定义调用每个参与者行为的步骤方法。

其他世界可以定义包含乘客比其他行为者。通过提供步骤方法的不同实现和世界类的其他方法，一种能够生产模拟，游戏，等等。

为了显示图形用户界面，一个亚军程序构造的世界中，增加了驾乘人员的，并调用世界上的表演方法。该方法使GUI启动。

该ActorWorld类有一个网格<演员>参数的构造函数。使用该构造，探索世界与网格而不是默认的10×10格等。

该ActorWorld有两种添加一个演员。

公共无效添加（位置禄，演员乘员）
公共无效添加（演员乘员）
没有一个位置参数的add方法在随机空位置增加了一个演员。

当添加演员的世界里，一定要使用ActorWorld类的add方法，而不是电网接口的put方法。 add方法调用演员的方法putSelfInGrid。如前所述，在putSelfInGrid方法将演员的引用，其网格和位置，并调用网格法放，给电网的引用演员。

删除方法删除指定位置的演员，并返回已被删除的演员。

公共演员删除（位置LOC）
图形用户界面，世界，和演员类之间的关系如图所示。需要注意的是GUI毫不知情的演员。它可以显示在任何世界居住者。相反，演员不认识的GUI。

余浩强 发表： Aug 21, 2015
666

林育新 发表： Aug 21, 2015
I listen, reply u can get the Chinese.

朱侨荣 发表： Aug 21, 2015
u

殷家康 发表：

Aug 21, 2015

666

戴旋 发表：

Aug 21, 2015

66666

徐欣 发表：

Aug 22, 2015

666

殷家康 发表：

Aug 21, 2015

话说Jumper那些问题中的情况是我们自己想怎么处理就怎么处理吧?

没有什么要求限制吧= =

邱永臣 发表：

Aug 21, 2015

Do whatever you like