AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 3: MultiGrid (GridWorld)

Part A: get **2 1/2 points**

+1/2 `grid.get(loc)`

+1 1/2 null case
- **+1/2** check if `grid` contents are null
- **+1/2** construct empty set if null (nongeneric okay)
- **+1/2** return empty set if null (lose this if add set to grid)

+1/2 return `grid` contents if not null

Part B: put **2 points**

+1/2 `get(loc)` (or `grid.get(loc)` with null check)

+1/2 add `obj` to accessed set (or empty set, as required)

+1/2 `grid.put(loc,`*updatedSet*`)` (in empty case)

+1/2 correct in all cases

Part C: getNeighbors **4 1/2 points**

+1/2 construct an `ArrayList` of `Object`s (must store in a variable)

+3 1/2 add neighbors to list
- **+1/2** access a neighboring location (e.g., `grid.getNeighbors(loc)` or `loc.getAdjacentLocation(`*dir*`)`)
- **+1 1/2** traverse sets from neighboring locations
  - **+1/2** correctly access a neighboring set (in context of loop)
  - **+1** access all neighboring sets
- **+1** traverse accessed set of neighbors
  - **+1/2** correctly access an element of set (in context of loop)
  - **+1/2** access all elements of set
- **+1/2** add neighbor object to neighbor list

+1/2 return neighbor list

Note: -1 usage error for accessing `occupantMap` in Parts (A) and (B)

© 2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AP® COMPUTER SCIENCE AB
2008 CANONICAL SOLUTIONS

Question 3: MultiGrid (GridWorld)

PART A:

```
public Set<Object> get(Location loc)
{
  Set<Object> objectsAt = grid.get(loc);
  if (objectsAt == null)
    objectsAt = new HashSet<Object>();

  return objectsAt;
}
```

PART B:

```
public void put(Location loc, Object obj)
{
  Set<Object> objectsAt = get(loc);
  objectsAt.add(obj);
  grid.put(loc, objectsAt);
}
```

PART C:

```
public ArrayList<Object> getNeighbors(Location loc)
{
  ArrayList<Set<Object>> neighborSets = grid.getNeighbors(loc);

  ArrayList<Object> neighbors = new ArrayList<Object>();
  for (Set<Object> nextSet : neighborSets)
  {
    for (Object nextObject : nextSet)
      neighbors.add(nextObject);
  }
  return neighbors;
}
```

ALTERNATE SOLUTION:

```
public ArrayList<Object> getNeighbors(Location loc)
{
  ArrayList<Object> neighbors = new ArrayList<Object>();
  for (int dir = 0; dir < 360; dir += 45)
  {
    Location nextLoc = loc.getAdjacentLocation(dir);
    if (grid.isValid(nextLoc))
    {
      for (Object nextObject : get(nextLoc))
        neighbors.add(nextObject);
    }
  }
  return neighbors;
}
```

© 2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AB3a,

(a) Complete the `UnboundedMultigrid` method `get` below.

```
/** @param loc a valid location in this grid
 *  @return a set of all objects at loc; an empty set, if no objects at loc
 *  Postcondition: the contents of this grid remain unchanged
 */
public Set<Object> get(Location loc)
{
    if (grid.get(loc) == null)
        return new HashSet<Object>();
    else
        return grid.get(loc);
}
```

GO ON TO THE NEXT PAGE.

©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

(b) Assume that the `UnboundedMultigrid` method `get` works as specified, regardless of what you wrote in part (a).

Complete the `UnboundedMultigrid` method `put` below.

```
/** Puts an object at a given location in this grid.
 *  Precondition: (1) loc is valid in this grid. (2) obj is not null.
 *  @param loc the location at which to put the object
 *  @param obj the new object to be added
 */
public void put(Location loc, Object obj)
{
    Set<Object> s = get(loc);
    s.add(obj);
    grid.put(loc, s);
}
```

Part (c) begins on page 18.

©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

(c) Assume that the `UnboundedMultigrid` methods `get` and `put` work as specified, regardless of what you wrote in parts (a) and (b).

Complete the `UnboundedMultigrid` method `getNeighbors` below.

```
/** Gets the neighboring occupants in all eight compass directions
 *  (north, northeast, east, southeast, south, southwest, west, and northwest).
 *  @param loc a location in this grid
 *         Precondition: loc is valid in this grid
 *  @return an array list of the objects in the occupied locations adjacent to loc in this grid
 */
public ArrayList<Object> getNeighbors(Location loc)
{
    ArrayList<Object> a = new ArrayList<Object>();
    for (Set<Object> s : grid.getNeighbors(loc))
    {
        for (Object obj : s)
        {
            a.add(obj);
        }
    }
    return a;
}
```


©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AB3b1

(a) Complete the `UnboundedMultigrid` method `get` below.

```
/** @param loc a valid location in this grid
 *  @return a set of all objects at loc; an empty set, if no objects at loc
 *  Postcondition: the contents of this grid remain unchanged
 */
public Set<Object> get(Location loc){
    Set<Object> temp = grid.get(loc);
    if(temp == null){
        return new Set<Object>();
    }
    else return temp;
}
```


©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

(b) Assume that the `UnboundedMultigrid` method `get` works as specified, regardless of what you wrote in part (a).

Complete the `UnboundedMultigrid` method `put` below.

```
/** Puts an object at a given location in this grid.
 *  Precondition: (1) loc is valid in this grid. (2) obj is not null.
 *  @param loc the location at which to put the object
 *  @param obj the new object to be added
 */
public void put(Location loc, Object obj){
        grid.get(loc).add(obj);
}
```

Part (c) begins on page 18.

©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

(c) Assume that the `UnboundedMultigrid` methods `get` and `put` work as specified, regardless of what you wrote in parts (a) and (b).

Complete the `UnboundedMultigrid` method `getNeighbors` below.

```
/** Gets the neighboring occupants in all eight compass directions
 *  (north, northeast, east, southeast, south, southwest, west, and northwest).
 *  @param loc a location in this grid
 *         Precondition: loc is valid in this grid
 *  @return an array list of the objects in the occupied locations adjacent to loc in this grid
 */
public ArrayList<Object> getNeighbors(Location loc){
    ArrayList<Object> neighbors = new ArrayList<Object>();

    for(int i = 0; i < 360; i += 45){
      Location ad; = loc.getAdjacentLocation(i);
    Set<Object> n = grid.get(ad));
      for(object o ; n){
          nieghbors.add(o);
      }
    }
    return neighbors;
}
```


©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

(a) Complete the `UnboundedMultigrid` method `get` below.

```
/** @param loc a valid location in this grid
 *  @return a set of all objects at loc; an empty set, if no objects at loc
 *  Postcondition: the contents of this grid remain unchanged
 */
public Set<Object> get(Location loc)
    {
      Set<Object> inLoc = new Set<Object>();
      if (loc == null)
        return inLoc;
      else
        return grid.get(loc);
    }
```


©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AB3c2

(b) Assume that the `UnboundedMultigrid` method `get` works as specified, regardless of what you wrote in part (a).

Complete the `UnboundedMultigrid` method `put` below.

```
/** Puts an object at a given location in this grid.
 *  Precondition: (1) loc is valid in this grid. (2) obj is not null.
 *  @param loc the location at which to put the object
 *  @param obj the new object to be added
 */
public void put(Location loc, Object obj)
    {
      if (loc == null)
        throw new NullPointerException("loc == null");
      if (obj == null)
        throw new NullPointerException("obj == null");
      grid.put(loc, obj);
    }
```

Part (c) begins on page 18.

©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AB3c3

(c) Assume that the `UnboundedMultigrid` methods `get` and `put` work as specified, regardless of what you wrote in parts (a) and (b).

Complete the `UnboundedMultigrid` method `getNeighbors` below.

```
/** Gets the neighboring occupants in all eight compass directions
 *  (north, northeast, east, southeast, south, southwest, west, and northwest).
 *  @param loc a location in this grid
 *         Precondition: loc is valid in this grid
 *  @return an array list of the objects in the occupied locations adjacent to loc in this grid
 */
public ArrayList<Object> getNeighbors(Location loc)
{
    ArrayList<Location> locs = new ArrayList<Location>();
    int d = Location.NORTH;
    for (int i = 0; i < Location.FULL_CIRCLE/Location.HALF_RIGHT; i++)
    {
        Location neighborLoc = loc.getAdjacentLocation(d);
        if (isValid(neighborLoc))
            locs.add(neighborLoc);
        d = d + Location.HALF_RIGHT;
    }
    return locs;
}
```

GO ON TO THE NEXT PAGE.

©2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

AP® COMPUTER SCIENCE AB
2008 SCORING COMMENTARY

Question 3

Overview

This question was based on the GridWorld case study and involved re-creating and extending the functionality of GridWorld classes. The idea of extending a grid so that multiple objects could be stored in a single location was described, and a `Multigrid` interface was provided to formalize this idea. Students were then asked to complete methods of an `UnboundedMultigrid` class that implements that interface. In part (a) they were required to implement the `get` method, which involved accessing and returning the set of objects stored at the specified location. In part (b) students had to implement the `put` method, which involved adding an object to the set at the specified location. In part (c) they were required to implement the `getNeighbors` method, which involved accessing all of the neighbor locations in the multigrid and collecting the objects stored in the neighboring sets.

Sample: AB3a
Score: 9

The student received the full 2½ points for part (a). There is a reference to `grid.get(loc)`. The student correctly compares `grid.get(loc)` to `null` and in the case it is `null` correctly constructs then returns an empty `HashSet`. In the case `grid.get(loc)` is not `null` the student correctly returns the grid contents at this location.

In part (b) the student earned the full 2 points. The student stores the result of `get(loc)` in `s`, correctly adds `obj` to the set, and puts the updated set back into the grid. The `put` is not necessary if the original set returned by `get(loc)` were not empty, but it causes no harm either. If the original set returned by `get(loc)` were empty, then putting the updated set into the grid is necessary. The student correctly handles the case where the original set is not empty and the case where the original set returned is empty. So the student earned each of the four ½ points.

In part (c) the student has a correct solution earning 4½ points. The student correctly constructs an `ArrayList` of `Object`s. The student then correctly uses a for-each loop to iterate over the sets adjacent to `loc` returned from `grid.getNeighbors(loc)`. For each set, the student correctly uses a for-each loop to iterate over the objects in the set. Finally the student correctly adds the objects in the adjacent sets to the `ArrayList a` and returns it.

Sample: AB3b
Score: 7

The student earned 2 points in part (a). The student calls `grid.get(loc)` and correctly compares the returned value to `null`. The student lost the ½ point for constructing the empty set since one cannot instantiate `Set`, which is an interface; however, the student earned the ½ point for returning the empty set in the case of `null`. Finally the student correctly returns the result from `grid.get(loc)` in the case where it is not `null`.

In part (b) the student earned only the second ½ point in the scoring guidelines. The student lost the first ½ point since `grid.get(loc)` is used rather than `get(loc)` written in part (a). The only way students can get credit for the call to `grid.get(loc)` is if they correctly test that it is not `null`. The student earned the ½ point for adding `obj` to the accessed set. Since there is no call to `grid.put()` and the student does not handle the case where the set returned from `get(loc)` is empty, the student lost the last two ½ points in the scoring guidelines.

© 2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.

Question 3 (continued)

In part (c) the student earned 4 of the 4½ points. The student correctly constructs an `ArrayList` of `Objects`. The outer for loop allows the student to access each of the eight locations adjacent to `loc`. Note that since `grid` is an `UnboundedGrid`, there is no need to ensure each of the adjacent locations is valid. The student lost ½ point for correctly accessing a neighboring set since the student uses `grid.get(adj)` to retrieve the set and does not test for `grid.get(adj)` being `null`. After this deduction, however, the student earned the full 1 point for accessing all the neighboring sets since the loop structure is correct. And the student correctly traverses each neighboring set using the for-each loop. The student adds the objects in the adjacent sets to the `ArrayList neighbors` and returns it.

The total score of 6½ points was rounded up to 7.

Sample: AB3c
Score: 3

The student earned 1½ points in part (a). The student earned the first ½ point for the reference to `grid.get(loc)`. The student lost the ½ point for checking if the grid contents are `null` since the code compares `loc` to `null` rather than `grid.get(loc)` to `null`. The student lost the ½ point for constructing the empty set due to the use of `Set`, which is an interface. However, the student earned the ½ point for returning the empty set.

The student earned no points in part (b). There is no reference to either `get(loc)` or `grid.get(loc)`. The student does not add `obj` to an accessed set. The `put` is incorrect since it puts the object and not an updated set into the grid. The student lost the last ½ point since there is no attempt to handle the empty case.

The student earned 1 point in part (c). The student lost the first ½ point because the `ArrayList locs` is created as an `ArrayList` of `Locations` instead of an `ArrayList` of `Objects`. The student earned the ½ point for accessing a neighboring location with `loc.getAdjacentLocation(d)`. The student lost the next 3 points from the scoring guidelines because there is no reference to sets in the adjacent locations. The student instead fills the `ArrayList` with valid adjacent locations. The student earned the last ½ point in the scoring guidelines for returning a neighbor list in the sense that it is a list of neighboring locations.

The total score of 2½ points was rounded up to 3.

© 2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.