AP® COMPUTER SCIENCE AB
2007 SCORING GUIDELINES

Question 2: Pair Matcher

Part A: constructor **5 points**

+1/2 `personMap =  new HashMap<Person, PriorityQueue<Pair>>();`

+1 iterate over `personList`
- **+1/2** access a `personList` element in loop body
- **+1/2** access all `personList` elements

+2 construct priority queue
- **+1/2** `new PriorityQueue<Pair>()`
- **+1 1/2** add pairs to priority queue
  - **+1/2** nested iteration through entire `PersonList`
  - **+1/2** construct `Pair` object containing 2 `Person`s
  - **+1/2** add pair to the priority queue (but not duplicate)

+1 1/2 Put priority queues into map
- **+1/2** put at least one priority queue into `personMap`
- **+1** put a unique priority queue for every person in `PersonList`

Part B: removeNumMatches **4 points**

+1/2 `personMap.get(p)`

+1/2 return null if `p` is not in `personMap`

+2 1/2 store persons in array
- **+1/2** `new Person[num]`
- **+1** remove and access
  - **+1/2** remove pair from front of queue
  - **+1/2** add `person2` from pair to array
- **+1** repeat to add exactly `num` persons to array

+1/2 return array of persons

AP® Computer Science AB
2007 Canonical Solutions

Question 2: Pair Matcher

PART A:

```
public PairMatcher(List<Person> personList)
{
    personMap = new HashMap<Person, PriorityQueue<Pair>>();
    for (Person p : personList) {
        PriorityQueue<Pair> queue = new PriorityQueue<Pair>();
        for (Person p1 : personList) {
            if (p != p1) {
                queue.add(new Pair(p, p1));
            }
        }
        personMap.put(p, queue);
    }
}
```

PART B:

```
public Person[] removeNumMatches(Person p, int num)
{
    PriorityQueue<Pair> queue = personMap.get(p);
    if (queue == null) {
        return null;
    }

    Person[] matches = new Person[num];
    for (int i = 0; i < num; i++) {
        matches[i] = queue.remove().getPerson2();
    }
    return matches;
}
```

AB2a

Complete the PairMatcher constructor below.

```
/** Initializes and fills personMap so that each Person in personList is a key,
 *  and the value associated with each key k is a PriorityQueue of Pair objects
 *  pairing k with all other Persons in personList
 *  @param personList a nonempty list of Person objects
 */
public PairMatcher(List<Person> personList)
{    personMap = new HashMap<Person, PriorityQueue<Pair>>();
     Iterator<Person> it = personList.iterator();
     While (it.hasNext())
     {  Person p = it.next(),

     PriorityQueue<Pair> pq = new PriorityQueue<Pair>();

     for (Person partner : personList)
     {   if (! partner.equals(p))
              pq.add (new Pair(p, partner));
     }

     personMap.put(p, pq);
 }
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

AB2a

(b) Write the `PairMatcher` method `removeNumMatches`. Method `removeNumMatches` removes the first `num` `Pair` objects from the priority queue associated with `p` in `personMap` and returns an array containing the second `Person` of each `Pair` that was removed. The `Person` objects in the returned array should be ordered by their compatibility with `Person p`. If `Person p` is not in the map, `null` is returned.

Complete method `removeNumMatches` below.

```
/** @param p the Person to be matched
 *  @param num the number of Person objects to remove
 *              Precondition: if p is in personMap, then num is > 0 and less than or equal to
 *                            the number of pairs in the priority queue associated with p
 *  @return an array of the num removed Person objects;
 *               null if p is not in personMap
 */
public Person[] removeNumMatches(Person p, int num)
{   if (! personMap.containsKey(p))
          return null;

    Person[] matches = new Person[num];
    PriorityQueue<Pair> pq = personMap.get(p);
    for (int i = 0; i < num; i++)
    {   Pair group = pq.remove();
        Person partner = group.getPerson2();

        matches[i] = partner;
    }

    return matches;
}
```

Complete the `PairMatcher` constructor below.

```
/** Initializes and fills personMap so that each Person in personList is a key,
 *  and the value associated with each key k is a PriorityQueue of Pair objects
 *  pairing k with all other Persons in personList
 *  @param personList a nonempty list of Person objects
 */
public PairMatcher(List<Person> personList)
{
  for (int i=0; i < personList.length; i++)
  {
    ListIterator iter = personList.listIterator();


      Person p = iter.next();
      Person temp = iter.remove ();
      PriorityQueue other = new PriorityQueue();
      while (iter.hasNext())
      {
        other.add (new Pair (p, iter.next()));
      }
       personMap.put ( p , other);
       iter.add (temp);
       personList = iter;
    }
}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

(b) Write the `PairMatcher` method `removeNumMatches`. Method `removeNumMatches` removes the first `num` `Pair` objects from the priority queue associated with `p` in `personMap` and returns an array containing the second `Person` of each `Pair` that was removed. The `Person` objects in the returned array should be ordered by their compatibility with `Person p`. If `Person p` is not in the map, `null` is returned.

Complete method `removeNumMatches` below.

```
/** @param p the Person to be matched
 *  @param num the number of Person objects to remove
 *            Precondition: if p is in personMap, then num is > 0 and less than or equal to
 *                          the number of pairs in the priority queue associated with p
 *  @return an array of the num removed Person objects;
 *            null if p is not in personMap
 */
public Person[] removeNumMatches(Person p, int num)
{
  if (!(personMap.containsKey(p))) return null;
  else
  {
    Person[] others = new Person[num];
    PriorityQue temp = new PriorityQue();
    temp = personMap.get(p);
    for (int i=0; i< num; i++)
    {
      others[i] = (Person) (temp.remove());
    }
    return others;
  }
}
```

GO ON TO THE NEXT PAGE.

AB2c

Complete the `PairMatcher` constructor below.

```
/** Initializes and fills personMap so that each Person in personList is a key,
 *   and the value associated with each key k is a PriorityQueue of Pair objects
 *   pairing k with all other Persons in personList
 *   @param personList a nonempty list of Person objects
 */
public PairMatcher(List<Person> personList)
        { Iterator itr2 = personList.iterator();
          Iterator itr = personList.iterator();

        while( itr2. hasNext())
          {
              personMap.put (itr.next(), new Pair(
                                itr2.next(), itr.next())

          }

        }
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

AB2c

(b) Write the `PairMatcher` method `removeNumMatches`. Method `removeNumMatches` removes the first `num` `Pair` objects from the priority queue associated with `p` in `personMap` and returns an array containing the second `Person` of each `Pair` that was removed. The `Person` objects in the returned array should be ordered by their compatibility with `Person p`. If `Person p` is not in the map, `null` is returned.

Complete method `removeNumMatches` below.

```
/** @param p the Person to be matched
 *  @param num the number of Person objects to remove
 *         Precondition: if p is in personMap, then num is > 0 and less than or equal to
 *                       the number of pairs in the priority queue associated with p
 *  @return an array of the num removed Person objects;
 *          null if p is not in personMap
 */
public Person[] removeNumMatches(Person p, int num)
```

```
    { ArrayList<Person> x = new ArrayList(
 Object obj=personMap.get(p)

 PriorityQueue stuff = (PriorityQueue) obj;
  While (! stuff.isEmpty())
      { x.add(stuff.pop()); 
      }

ArrayList

   Person[] people = new Person[x.size()-1]
        int n = 0
        While (people[people.length] == null)
          { people[n] = x.get(n)
          n = n + 1;
          }

    return people;
   }
```

GO ON TO THE NEXT PAGE.

Question 2

Overview

This question centered on abstraction, interacting classes, and data structure use. Students were given two black-box classes: a `Person` class for representing a person and a `Pair` class for representing pairs of persons. In addition, the skeleton of a `PairMatcher` class was provided, which contained a `Map`, mapping a `Person` object to a priority queue of `Pair` objects. In part (a) students were required to implement the constructor for the `PairMatcher` class, which involved initializing the `Map` field, traversing a `List` of `Persons`, adding a `Map` entry for each `Person`, and constructing `Pairs` and inserting them into the `Map`. In part (b) they were required to implement the `removeNumMatches` method, which removes a specified number of `Pairs` from the `Map` (associated with a specific `Person`) and collects them in an array.

Sample: AB2a
Score: 9

The student correctly answers all of part (a). It is worth noting that the student uses an iterator for the outer loop and a for each as the nested loop. The student earned 5 points for part (a).

The student correctly answers all of part (b). Note that the student tested whether `p` is a key in `personMap` using the `containsKey` method. The student earned 4 points for part (b).

Sample: AB2b
Score: 6½

The solution for part (a) lost the ½ point for not instantiating `personMap` and the ½ point for incorrectly instantiating a `PriorityQueue` (missing type parameter). The student correctly removed the key from the front of the list via a `ListIterator` prior to matching it with the rest of the list, so there are no duplicates. Then after the iterator has reached the end of the list, the key is added back to the list. Note that this can only be done with a `ListIterator` (and not an `Iterator`, which does not have an `add` method). Since the outer loop index is not being used to access the key elements, this correctly accesses every key and pairs it with all other list elements. However, there is extraneous code that is incorrect (`personList = iter`), so there was a 1-point usage penalty applied to this solution. The student earned 3 points for part (a).

In part (b) the student lost the ½ point for adding the pair from the priority queue rather than the second person of the pair. All other parts of the solution are correct, so the student earned 3½ points for part (b).

Question 2 (continued)

Sample: AB2c
Score: 2

In part (a) the student earned a ½ point for accessing at least one element of `personList` and attempting to use it as a key in `personMap` or an element of a `Pair` construction, and a ½ point for correct construction of a `Pair`. Because the solution executes the iterator `next` method twice within loop body, the outer loop will attempt to access objects beyond the end of the list. There is no instantiation of a priority, no nested iteration, and the student attempts to put the `Person` key with a `Pair`. The student earned 1 point for part (a).

In part (b) the student earned a ½ point for getting the `PriorityQueue` associated with `p` from `personMap` and a ½ point for returning an array of `Person`. There is no test for whether `p` is a key in `personMap`. The student uses the wrong method name (`pop`) to remove the elements from the priority queue and removes all of the elements into an `ArrayList`. As a result, the array of `Person` is instantiated with the wrong size. The student adds the pair from the array list rather than the second person of the pair. The loop condition is incorrect, and the loop body adds every other element to the array. The student earned 1 point for part (b).