Northeastern University

Lecture 15: Mid-term Exam Review

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Format

- The exam will be 5-7 problems, with some problems having multiple sub-questions
- It is a closed-book exam. You need to bring in your computer (with a charger) to the class.
- No calculators, books, phones, or anything else.

Format

- Kinds of questions to expect:
  - » Explain OOP concepts and terminologies
  - » Explain a program or part of a program
  - » Write your own code
  - » Fix incorrect code / find bugs in code
  - » Fill in the blank (in a program)
  - » Short answer
  - » Multiple choice questions

Format

- Essentially, everything we've covered so far in the semester, including:
  - » Classes, Objects and Methods
  - » Generics, Inheritance and Interfaces
  - » Bags and Implementations - Array
  - » Bags and Implementations - Linked Data
  - » The Efficiency of Algorithms

Review Exercises

- The following slides contain exercises that will help you prepare for the exam.
- The exercises give you an idea of the style of questions to expect as well as the complexity.

Type 1: Terminology Explanation

Exercise

- What is the meaning of Inheritance? Please give an example of it.

Answer

- Inheritance allows you to define a very general class and then later define more specialized classes that add some new details to the existing class definition.
- Example:

```
public class Instrument {
    private String name;
    public Instrument(){
        …
    }
    …
}
```

```
public class Guitar extends Instrument {
    private String Brand;
    public Guitar(){
        super();
        …
    }
    …
}
```

Exercise

- What is the meaning of Interface? Please give an example of it.

Answer

- A Java Interface is a program component that contains the headings for a number of public methods.
- It does not declare any constructors for a class, instance variables and any complete method definitions (i.e., methods cannot have bodies).
- Example:

```
public interface Measurable {
  public double getPerimeter();
  public double getArea();
}
```

```
public class Rectangle implements Measurable {
  ...
  public double getPremeter(){
    return 2 * (myWidth + myHeight);
  }

  public double getArea(){
    return myWidth * myHeight;
  }
}
```

Exercise

- What is the meaning of Abstract Data Type (ADT)? In terms of ADT, what does Data Structure mean?

Answer

- Abstract Data Type (ADT)
  - » A specification that describes a data set and the operations on that data
  - » Does not indicate how to store the data or how to implement the operations
  - » Independent of any programming language
- Data Structure
  - » An implementation of an ADT within a programming language

What are other important concepts?

- Encapsulation
- Polymorphism
- Overloading vs. Overriding
- Interface vs. Abstract Class
- Generics
- Casting (Implicit and Explicit)

Type 2: Explain Results or Part of the Program

Exercise

- What output will be produced by the following code?

```
public class Overload {
    public static void main(String[] args){
        double average1 = Overload.getAverage(4.0, 50.0);
        double average2 = Overload.getAverage(1.0, 2.0, 3.0);
        char average3 = Overload.getAverage('a','c');

        System.out.println("average1 = " + average1);
        System.out.println("average2 = " + average2);
        System.out.println("average3 = " + average3);
    }
    public static double getAverage(double first, double second){
        return (first + second)/2.0;
    }
    public static double getAverage(double first, double second, double third){
        return (first + second + third)/3.0;
    }
    public static char getAverage(char first, char second){
        return (char)(((int)first + (int)second)/2);
    }
}
```

Answer

- average1 = 27.0
- average2 = 2.0
- average3 = b

Question

- Following is a code snippet of the Bag Implementation that uses Arrays.

```
public boolean remove(T anEntry)
{
   checkIntegrity();
   int index = getIndexOf(anEntry);
   T result = removeEntry(index);

   return anEntry.equals(result);
}
```

- As enclosed in a red color, can the `return` statement in the `remove` method to be written as follows?

```
a. return result.equals(anEntry);
b. return result != null;
```

Answer

a. `return result.equals(anEntry);` 

b. `Return result != null;` 

- For (a), the `result` might be `null` such that you will have an error because you try to access `null`'s `equals()` method.

Exercise

- The following algorithm discovers whether an array contains duplicate entries within its first n elements. What is the Big Oh of this algorithm in the worst case?

```
Algorithm hasDuplicates(array, n)
for index = 0 to n - 2
   for rest = index + 1 to n - 1
      if (array[index] equals array[rest])
         return true
return false
```

Answer

- Various values of index vs. inner loop iterations

index	Inner Loop Iterations
0	$n-1$
1	$n-2$
2	$n-3$
. . .	. . .
$n-2$	1

- The maximum number of times the inner loop executes is $1+2+...+n-1$, which is $n(n-1)/2$
- The algorithm is $O(n^2)$

Type 3: True/False Questions

Exercise

- Please answer “true” or “false” to the following questions? If you answer “false”, please also explain why.

Exercise

Q: A class can implement more than one interface by using extends.

Answer

- False. A Class can implement more than one interfaces by using “implements”, not “extends”.
- For example:

```
public interface Mammal{
    public void eat();
    public void travel();
}

public interface Human{
    public void study();
}

public class Kids implements Mammal, Human{
    public void study(){...}
    public void eat(){...}
    public void travel(){...}
}
```

Exercise

Q: When comparing two quantities of a class type to see whether they are “equal,” we should use ==.

Answer

- False
- We normally use the method equals when testing two objects to see whether they are “equal.” The only time you would use == is if you wanted to see whether the objects were in the same place in memory, that is , were identical.

Exercise

Q: The outer `LinkedBag` class can access the data fields of the inner `Node` class directly without using `set` and `get` methods.

Answer

- True

Exercise

Q: A chain requires less memory than an array of the same length.

Answer

- False
- A chain needs more memory to store references to linked nodes.

Type 4: Short Answer

Exercise

- Why is it better to implement the `add` operation in a collection before implementing the `remove` operation?

Answer

- You cannot test `remove` functionality until you have written and tested `add` functionality.

Exercise

- Should `Node` be a public class? Explain.

Answer

- No
- The details of implementation, whether array based or linked list based, should be hidden from the client.

Exercise

- If you have a $O(\log n)$ algorithm running, what happens when you double the size of your problem?

Answer

- The change is negligible.
- You simply add one step.

$$O(\log 2n) = O(\log 2 + \log n) = O(\log n)$$

Type 5: Multiple Choice

Exercise

- What is the worst-case time complexity for searching a linked-based bag ADT for a particular entry?
  - $O(n)$
  - $O(1)$
  - $O(n^2)$
  - `negligible`

Answer

- What is the worst-case time complexity for searching a linked-based bag ADT for a particular entry?
  - $O(n)$ **
  - $O(1)$
  - $O(n^2)$
  - `negligible`

Exercise

- Which one of the following Java statements allocates an array in the bag constructor causing a compiler warning for an unchecked operation? Assume capacity is an integer.
  - `bag = (T[ ]) new Object[capacity];`
  - `bag = new T[capacity];`
  - `bag = new Object[capacity];`
  - `bag = new (T[ ]) Object[capacity];`

Answer

Type safety: Unchecked cast from Object[] to T[]

- `bag = (T[ ]) new Object[capacity]; **`
- `bag = new T[capacity];`
- `bag = new Object[capacity];`
- `bag = new (T[ ]) Object[capacity];`

Exercise

- Which instruction suppresses an unchecked-cast warning from the compiler?
  - `@SuppressUnchecked()`
  - `@Warning("suppress unchecked")`
  - `@SuppressWarnings("unchecked")`
  - `@SuppressUncheckedWarnings()`

Answer

- Which instruction suppresses an unchecked-cast warning from the compiler?
  - `@SuppressUnchecked()`
  - `@Warning("suppress unchecked")`
  - `@SuppressWarnings("unchecked")` **
  - `@SuppressUncheckedWarnings()`

Exercise

- Decrementing the private class variable *`numberOfEntries`* in a bag
  - causes the last entry to be ignored
  - causes an IllegalState exception to be thrown
  - destroys the integrity of the bag container
  - is a security problem

Answer

- Decrementing the private class variable *`numberOfEntries`* in a bag
  - causes the last entry to be ignored **
  - causes an IllegalState exception to be thrown
  - destroys the integrity of the bag container
  - is a security problem

Type 6: Write Your Own Code

Exercise

- Write a Java interface that specifies and declares methods for a class of `students`.

Answer

```
public interface StudentInterface {
  public void setStudent(Name studentName, String studentId);
  public void setName(Name studentName);
  public Name getName();
  public void setId(String studentId);
  public String getId();
  public String toString();
}
```

StudentInterface.java

Answer

```
public class Name {
  private String firstName;
  private String lastName;

  public Name(String first, String last){
    firstName = first;
    lastName = last;
  }

  public String toString(){
    return lastName + "," + firstName;
  }
}
```

Name.java

Exercise

- Begin the definition of a class that implements the interface that you wrote in answer to the previous question.
- Include data fields, a constructor, and at least one method definition.

Answer

```
public class Student implements StudentInterface{
  private Name fullName;
  private String id;

  public Student(){
    fullName = new Name("","");
    id = "";
  }

  public Student(Name studentName,
                 String studentId){
    fullName = studentName;
    id = studentId;
  }

  public void setStudent(Name studentName,
                         String studentId){
    setName(studentName);
    setId(studentId);
  }

  public void setName(Name studentName){
    fullName = studentName;
  }

  public Name getName(){
    return fullName;
  }

  public void setId(String studentId){
    id = studentId;
  }

  public String getId(){
    return id;
  }

  public String toString(){
    return id + " " + fullName.toString();
  }

} // End of Student class
```

Student.java

Answer

```
public class StudentTest {
  public static void main(String[] args){

    Name student1Name = new Name("Richard", "Anderson");
    String student1Id = "W123456789";

    Student student1 = new Student(student1Name, student1Id);

    System.out.println(student1.getId() + " " + student1.getName());

    //...
    //You can create another student and use setName() and setId()
  }
}
```

StudentTest.java