Exam 2 Review

Lecture 14

Format

- The exam will be in two parts…

1. Questions via Blackboard (in class)
   - T/F, multiple choice/answer, output, legal code?

2. Take-Home Coding via GitLab+Eclipse (think PA, sans peeps)
   - JUnit tests
   - Individual methods, classes, **`main`**

- For #1 you are allowed a single 8.5x11" piece of paper with whatever notes you wish
  - You cannot use other resources (e.g. prior programs, Google, peeps)
  - Bring your laptop charged!

Content (1)

Everything we've covered all semester, including…

- Anything COMP1000
  - Command-line arguments
- Classes and Objects
  - UML
  - Static vs Instance, Visibility, **`final`** (x3)
  - Packages
  - Wrappers, **`ArrayList`**, **`StringBuilder`**/**`StringBuffer`**
  - String interning
- Inheritance and Polymorphism
  - Terms (+ abstraction, encapsulation)
  - **`this`**, **`super`**
  - Constructor chaining, dynamic binding
  - Overriding vs. overloading, annotations
  - Concrete vs. **`abstract`** classes/methods
  - Explicit vs. implicit casting
  - **`instanceof`**, **`equals`**, **`toString`**

Content (2)

Everything we've covered all semester, including…

- JavaFX
  - Big-picture ideas
- Abstract Classes/Interfaces
- Generics
- Using JCF data structures
- Recursion

- Non-exhaustive notes follow…

Abstract Classes/Interfaces

- Multiple vs. Single Inheritance
- Code/UML: extends/implements
- Abstract methods vs. classes
- Lambda (be able to interpret)
- Why one vs. other

<G>enerics

- Goals (general code vs. early errors)
- Primitives vs. Types
- Classes, Interfaces, Methods
- Definition (extends); Wildcard (extends/super)
- Type erasure

Using JCF Data Structures

- What are data structures
- Collection vs. Map
- Iterator
- Why choose concrete implementations given a use case

Recursion

- Not necessary, overhead
- Code -> output

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

Question

- Which of the following is legal:
  A. interface A { void foo() {}; }
  B. abstract interface A { void foo() {}; }
  C. interface A { foo(); }
  D. interface A { void foo(); }

Solution

- Which of the following is legal:
  A. interface A { void foo() {}; }
  B. abstract interface A { void foo() {}; }
  C. interface A { foo(); }
  D. interface A { void foo(); }

Question

- Suppose A is an abstract class, B is a concrete subclass of A, and both A and B have a default constructor. Which of the following is legal?
  A. A a = new A();
  B. A a = new A(new B());
  C. B b = new A(new B());
  D. A a = new B();
  E. B b = new A();
  F. None of the above is legal

Solution

- Suppose A is an abstract class, B is a concrete subclass of A, and both A and B have a default constructor. Which of the following is legal?
  A. A a = new A();
  B. A a = new A(new B());
  C. B b = new A(new B());
  D. A a = new B();
  E. B b = new A();
  F. None of the above is legal

Question

- What is the output of the following code?

```
public class Test {
  public static void main(String[] args) {
    new A().print();
    new B().print();
  }
}

class B extends A {
  public String getInfo() {
    return "Bush";
  }
}
```

```
class A {
 public String getInfo() {
   return "Hand";
 }

 public void print() {
   System.out.println(getInfo());
 }
}
```

Solution

- What is the output of the following code?

```
public class Test {
  public static void main(String[] args) {
    new A().print();
    new B().print();
  }
}

class B extends A {
  public String getInfo() {
    return "Bush";
  }
}
```

```
Hand
Bush
```

```
class A {
 public String getInfo() {
   return "Hand";
 }

 public void print() {
   System.out.println(getInfo());
 }
}
```

Question

```
public static void m(LinkedList<? super Number>) {}
```

- Which of the following will compile?
  - A. `m(new LinkedList<Object>());`
  - B. `m(new LinkedList<Number>());`
  - C. `m(new LinkedList<Integer>());`
  - D. `m(new LinkedList<String>());`

Solution

```
public static void m(LinkedList<? super Number>) {}
```

- Which of the following will compile?
  - **A. `m(new LinkedList<Object>());`**
  - **B. `m(new LinkedList<Number>());`**
  - C. `m(new LinkedList<Integer>());`
  - D. `m(new LinkedList<String>());`

Question

- Will the following code compile?

```
public class Test implements
      Comparable<Integer>, Comparable<Double> {
   public int compareTo(Integer o) { return 0; }
   public int compareTo(Double o) { return 0; }
}
```

Solution

- Will the following code compile?
  - **NO (type erasure)**

```
public class Test implements
        Comparable<Integer>, Comparable<Double> {
    public int compareTo(Integer o) { return 0; }
    public int compareTo(Double o) { return 0; }
}
```

Question

- What is the output of the following code?

```
Set<Integer> s = new HashSet<>();
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next());
}
```

Question

- What is the output of the following code?

```
Set<Integer> s = new HashSet<>();
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next()); // ???
}
```

Question

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>();
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next());
}
```

Solution

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>();
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next()); // 1 2 3
}
```

Question

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>(
   (e1,e2)->e1.compareTo(e2));
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next());
}
```

Solution

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>(
   (e1,e2)->e1.compareTo(e2));
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next()); // 1 2 3
}
```

Question

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>(
   (e1,e2)->e2.compareTo(e1));
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next());
}
```

Solution

- What is the output of the following code?

```
Set<Integer> s = new TreeSet<>(
   (e1,e2)->e2.compareTo(e1));
s.add(3);
s.add(2);
s.add(1);
s.add(2);
s.add(3);

Iterator<Integer> i = s.iterator();
while (i.hasNext()) {
   System.out.printf("%d ", i.next()); // 3 2 1
}
```

Question

- A friend is writing a single-user game of Poker and needs a collection for their card objects. Would you recommend **`ArrayList`** or **`Vector`**? Why?

Solution

- ArrayList

Question

- Predict the output of running the following code

```
public static int s(int[] a, int x, int low, int high) {
      if (low > high) return -1;
      int mid = (low + high)/2;
      if (a[mid] == x) return mid;
      else if (a[mid] < x)
        return s(a, x, mid+1, high);
      else
        return s(a, x, low, mid-1);
  }


public static void main(String[] args) {
     int[] a = {1, 2, 4, 6, 7, 10, 11};
     System.out.println(s(a, 1, 0, a.length-1));
     System.out.println(s(a, 2, 0, a.length-1));
     System.out.println(s(a, 3, 0, a.length-1));
     System.out.println(s(a, 4, 0, a.length-1));
     System.out.println(s(a, 5, 0, a.length-1));
     System.out.println(s(a, 6, 0, a.length-1));
}
```

Solution

- Predict the output of running the following code

```
public static int s(int[] a, int x, int low, int high) {
      if (low > high) return -1;
      int mid = (low + high)/2;
      if (a[mid] == x) return mid;
      else if (a[mid] < x)
        return s(a, x, mid+1, high);
      else
        return s(a, x, low, mid-1);
  }


public static void main(String[] args) {
     int[] a = {1, 2, 4, 6, 7, 10, 11};
     System.out.println(s(a, 1, 0, a.length-1)); // 0
     System.out.println(s(a, 2, 0, a.length-1)); // 1
     System.out.println(s(a, 3, 0, a.length-1)); // -1
     System.out.println(s(a, 4, 0, a.length-1)); // 2
     System.out.println(s(a, 5, 0, a.length-1)); // -1
     System.out.println(s(a, 6, 0, a.length-1)); // 3
}
```

https://en.wikipedia.org/wiki/Binary_search_algorithm