Exam Info

- Prelim 1: 7:30–9:00PM, Tuesday, Nov 9th, Olin 155 (Last name starts with A-Lewis) and Olin 255 (Last name starts with Li-Z)
- Look at the previous Prelims
- Arrive early! Helps reducing stress

Topics

- Recursion
- Class Hierarchy
- Abstract
- Arrays
- Loops
- Invariants

Recursive Function 4 Principles

- 1. Write the precise specification

```java
/** = n, but with its digits reversed. 
 * Precondition: n >= 0.
 * e.g. n = 135720, value is “027531”.
 * e.g. n = 12345, value is “54321”.
 * e.g. n = 7, value is “7”.
 * e.g. n = 0, value is “0”.
 */
public static String rev(int n) {
    // base case:  
    // (n has only one digit)

    // recursive case: 
    // (n has at least two digits)

    return a String
}
```
Recursive Function 4 Principles

• 1. Write the precise specification

• 2. Base Case

• 3. Progress
  – Recursive call, the argument is “smaller than” the parameter. Ensures base case will be reached (which terminates the recursion)

• 4. Recursive case

Let’s review some type issues

What is the type of?

• 42
• "" + 42:
• ‘a’ + ‘b’
• ‘b’ + ”anana"
• ‘b’ + ‘a’ + ”nana"
• ‘b’ + (‘a’ + ”nana”)
• ”” + ‘b’ + ‘a’ + ”nana"

Recursive Function 4 Principles

/** = n, but with its digits reversed. 
  Precondition: n >= 0.
  e.g. n = 135720, value is “027531”.
  e.g. n = 12345, value is “54321”.
  e.g. n = 7, value is “7”.
  e.g. n = 0, value is “0”*/
public static String rev(int n) {
  // base case:
  // (n has only one digit)
  if (n < 10)
    return String.valueOf(n);
  // recursive case:
  // (n has at least two digits)
  return (n%10) + rev(n/10);
}
/** = the reverse of s. */
public static String rev(String s) {
    if (s.length() <= 1) return s; // base case

    // { s has at least two chars }
    int k = s.length() - 1;
    return s.charAt(k) +
            rev(s.substring(1, k)) +
            s.charAt(0); // recursive case
}

Do this one using this idea:
To reverse a string that contains at least 2 chars, switch first and last ones and reverse the middle.

(Fall’05) Question 4 (30 points) For each pair of statements below, write the value of d after execution. If the statements lead to an error, write “BAD” and briefly explain the error. (The question continues on the next page.)

Documentary e =
    new Short("Man on Wire", 5, "Bio");
boolean d =
    "Short Doc".equals(e.DocumentaryType());

(Fall’05) Question 4 (30 points) For each pair of statements below, write the value of d after execution. If the statements lead to an error, write “BAD” and briefly explain the error. (The question continues on the next page.)

Documentary e =
    new Short("Man on Wire", 5, "Bio");
boolean d =
    "Short Doc".equals(e.DocumentaryType());

True. method equals here is from the string object.
2. Movie c = new Documentary(null, 3, "Carter Peace Center");
   int d = c.popularity();

   public class Movie {
      private String title; // title of movie
      private int length; // length in minutes

      public Movie(String t, int l) {
         title = t; length = l;
      }

      public String getTitle() { return title; }
      public int getLength() { return length; }

      /** = the popularity: shorter means more popular */
      public int popularity() {
         return 300 - length;
      }
   }

   public class Documentary extends Movie {
      private String topic; // …

      public Documentary(String t, int n, String p) {
         super(t, n);
         topic = p;
      }

      /** = "Documentary" */
      public String DocumentaryType() {
         return "Documentary" ;
      }

      /** = popularity of this instance */
      public int popularity() {
         return 200 - getLength();
      }
   }

   public class Short extends Documentary {
      public Short(String t, int n, String p) {
         super(t, n, p);
      }

      /** displays acknowledgement */
      public String showAck() {
         return "We thank our director";
      }

      /** = "Short Doc" */
      public String DocumentaryType() {
         return "Short Doc";
      }
   }
3. Short b= (Short) (new Documentary("", 2, "WMD"));
   int d= b.DocumentaryType().length();
   Movie Documentary Trailer
   Short
   • From documentary, can go (cast) up and back down to documentary.
   • Think what would happen for the call b.showAck().
   • Answer: BAD

4. Movie a= (Movie) (new Trailer("Harry Potter"));
   int d= a.popularity();
   Movie Documentary Trailer
   Short
   • The cast is legal!
   • Method popularity() from Trailer is called (inherited from Trailer)

5. Movie f= new Short("War", 1, "Vietnam");
   char d= f.DocumentaryType().charAt(1);
   The methods that can be called are determined by the apparent type:
   Only components in the apparent class (and above)!!!
   f.DocumentaryType() is illegal. Syntax error.
   Answer: BAD

Recap: equals(Object ob)
   • In class Object
     – b.equals(d) is the same as b == d
     • Unless b == null (why?)
   • Most of the time, we want to use equals to compare fields. We need to override this method for this purpose
(Fall’05) Question 4 (24 points). (a) Write an instance method equals (Object obj) for class Documentary

```java
public class Documentary extends Movie {
    /** = “obj is a Documentary with the same values in its fields as this Documentary” */
    public boolean equals(Object obj) {
        return false;
    }
}
```

Don’t forget to cast. This is a legal cast. (Why?)

```java
public class Documentary extends Movie {
    /** = “obj is a Documentary with the same values in its fields as this Documentary” */
    public boolean equals(Object obj) {
        Documentary docObj = (Documentary) obj;
        return getTitle().equals(docObj.getTitle()) && getLength() == docObj.getLength() && topic.equals(docObj.topic);
    }
}
```

- Abstract Classes
Let’s capture the essence of animals

```java
/** representation of an animal */
public class Animal {
    private int birthDate; // animal’s birth date
    private String predator; // predator of this animal
    private String prey; // class of animals this hunts
    // Move the animal to direction...
    public void move(...) {
        //...;
    }
    // Make the animal eat...
    public void eat(...) {
        //...;
    }
}
```

Problems

- Animal is an abstract concept
  - Creating an abstract animal doesn’t make sense in the real world
  - Dogs, cats, snakes, birds, lizards, all of which are animals, must have a way to eat so as to get energy to move
- However...
  - Class Animal allows us to create a UFA (unidentified flying animal), i.e. instance of Animal
  - If we extend the class to create a real animal, nothing prevent us from creating a horse that doesn’t move or eat.

Solutions

- How to prevent one from creating a UFA?
  - Make class Animal abstract
  - Class cannot be instantiated
  - How? Put in keyword abstract
- How to prevent creation paralyzed dogs or starving sharks?
  - Make the methods move and eat abstract
  - Method must be overridden
  - How? Put in keyword abstract and replace the body with ";"

Making things abstract

```java
/** representation of an animal */
public abstract class Animal {
    private int birthDate; // animal’s birth date
    private String predator; // animal’s predator
    private String prey; // What animal hunts
    // Move the animal move in direction...
    public abstract void move(...) {
        //...;
    }
    // Make the animal eat...
    public abstract void eat(...) {
        //...;
    }
}
```

Arrays

Array: object

Can hold a fixed number of values of the same type.
The type of the array:
- int[]
- String[]
- Integer[]

Basic form of a declaration: int[] x
- Does not create array, it only declares x, x’s initial value is null.

Array creation: new int[4]
Array assignment: int[] y = new int[4]
Elements of array are numbered: 0, 1, 2, …, x.length-1;

7
Array: length

Array length: an instance field of the array. This is why we write x.length, not x.length( ).
Length field is final: cannot be changed. Length remains the same once the array has been created.

The length is not part of the array type. The type is int[].
An array variable can be assigned arrays of different lengths.

```
int[] x;
x = new int[4];
x = new int[32];
```

Array initializers

Instead of
```
int[] c = new int[5];
```
Use an array initializer:
```
int[] c = new int[] {5, 4, 7, 6, 5};
```

Question 3 (20 points)

a) Consider the program segment below. Draw all variables (with their respective values) and objects created by execution of this program segment.

```
int[] z = new int[] {3, 2, 1};
String[] s = new String[] {"Hi"};
z = new int[2];
```
b) Give an expression to reference the second element of z.
c) What is the result of the expression s[1].length() after the execution of the code above?
d) Give the declaration of a single variable v to store the values "1" and "Hi" at the same time.
Note on ranges.

2.5 contains 2, 3, 4, 5. It contains 5+1 – 2 = 4 values
2.4 contains 2, 3, 4. It contains 4+1 – 2 = 4 values
2.3 contains 2, 3. It contains 3+1 – 2 = 4 values
2.2 contains 2. It contains 2+1 – 2 = 1 values

The number of values in m..n is n+1 – m.
2.1 contains . It contains 1+1 – 2 = 0 values
3.1 contains . This is an invalid range.

In the notation m..n, we require always, without saying it, that m <= n + 1.
If m = n + 1, the range has 0 values.

Invariants

- **Assertions**: true-false statements (comments) asserting your beliefs about (the current state of) your program.

    // x is the sum of 1..n <- asserts a specific relationship between x and n

- **Invariant**: an assertion about the variables that is true before and after each iteration (execution of the repetend).

Finding an invariant

// Store in double variable v the sum // 1/1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n

```c
v = 0;
// invariant: v = sum of 1/i for i in 1..k-1
for (int k = 1; k <= n; k = k + 1) {
    Process k;
}
// v = 1/1 + 1/2 + ... + 1/n
```

Command to do something and equivalent postcondition

What is the invariant?

1 2 3 ... k-1 k k+1 ... n