/** = 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) 
    // recursive case: 
    // {n has at least two digits} 
    
}
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

• 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

public static String rev(int n) {
    // base case:
    if (n < 10)
        return “” + n;
    // recursive case:
    return (n%10) + rev(n/10);
}

/** = 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) {
    if (n < 10)
        return “” + n;
    // recursive case:
    return (n%10) + rev(n/10);
}

/** = the reverse of s.*/
public static String rev(String s) {
    if (s.length() <= 1)  // base case
        return s;
    // { s has at least two chars }
    int k= s.length()-1;
    return s.charAt(k) +  // recursive case
           rev(s.substring(1,k)) +
           s.charAt(0);  

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.
public class Trailer {
    // title of movie
    private String title; // title of trailer
    private int length; // length in minutes
    // constructor: a trailer of movie t.
    public Trailer(String t) {
        title = t; length = 1; // trailer one minute long.
    }
    // displays acknowledgement
    public String showAck() {
        return "We thank our director;"
    }
}

public class Documentary extends Movie {
    // title of movie
    private String title; // title of documentary
    private int length; // length in minutes
    private int popularity; // the popularity:
    // constructor: instance with title t, length n, and topic p
    public Documentary(String t, int n, String p) {
        super(t, n, p); // super constructor
    }
    // displays acknowledgement
    public String showAck() {
        return "We thank our director;"
    }
    // equals method
    boolean equals(Documentary d) {
        return title.equals(d.DocumentaryType());
    }
}

// Question 4 (30 points)
(Fall '05) 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());

2. Movie d =
   new Documentary(null, 3, "Carter Peace Center");
   int d = c.popularity();

20 17 15 13 14 16 18

QUESTION: Which method is called by Animal.toString()?
A. the one in the hidden partition for Object of a0
B. the one in partition Animal of a0
C. the one in partition Cat of a0
D. None of these

the class hierarchy:

Object
Animal
Cat
String, int
getNoisel()
toString()
2. Movie c =
   new Documentary(null, 3, "Carter Peace Center");
   int d = c.popularity();

   • What is the apparent class?
   • Answer: 197. method popularity of class Documentary is called

3. Short b = (Short)(new Documentary("", 2, "WMD");
   int d = b.DocumentaryType().length();

   • 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();

   • The cast is legal!
   • Which popularity() method is called?
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

\[
\text{public boolean equals(Object obj) \{}
\]
\[
  \text{if} \ (!\text{obj instanceof Documentary}) \{}
  \}
\[
\text{return false;}
\]
public class Documentary extends Movie {
    /** = "obj is a Documentary with the same values
     in its fields as this Documentary" */
    public boolean equals(Object obj) {
        if (!(obj instanceof Documentary) {
            return false;
        }
        Documentary docObj = (Documentary) obj;
        Don't forget to cast.
        This is a legal cast. (Why?)
    }

    public class Documentary extends Movie {
    /** = "obj is a Documentary with the same values
     in its fields as this Documentary" */
    public boolean equals(Object obj) {
        if (!(obj instanceof Documentary) {
            return false;
        }
        Documentary docObj = (Documentary) obj;
        return getTitle().equals(docObj.getTitle()) &&
               getLength() == docObj.getLength() &&
               topic.equals(docObj.topic);
    }

    Let’s capture the essence of animals
    /** 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 ";"
Array: object

Can hold a fixed number of values of the same type.

The type of the array:
- int
- String
- Integer

Array creation: new int[4]
Assignment: int[] t = new int[4]

Elements of array are numbered: 0, 1, 2, ..., x.length–1;

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.

An array variable can be assigned arrays of different lengths.

Arrays

<table>
<thead>
<tr>
<th>int[] x</th>
<th>x = new int[4]</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[0] = 0; x[1] = 0; x[2] = 0; x[3] = 0;</td>
<td>Create array object of length 4, store its name in x</td>
<td></td>
</tr>
<tr>
<td>x[2] = 5; x[3] = -4;</td>
<td>Assign 5 to array element 2 and -4 to array element 0</td>
<td></td>
</tr>
<tr>
<td>x[0] is a reference to element number 2 of array x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int k = 3; x[k] = 2 * x[0]; x[k-1] = 6;</td>
<td>Assign 2*x[0], i.e. -8, to x[3]</td>
<td>Assign 6 to x[2]</td>
</tr>
</tbody>
</table>

Difference between Vector and array

Declaration:
- int[] a;
- Vector v;

Elements of a: int values
Elements of v: any Objects

Creation:
- a = new int[n];
- v = new Vector();

Array always has n elements. Number of elements can change
Reference:
- a[0] v.get(0)

Change element:
- a[0] = e1;
- v.set(0, e1);

Array locations a[0], a[1], ... in successive locations in memory. Access takes same time no matter which one you reference.
Elements all the same type (a primitive type or class type)

Elements of any Object type (but not a primitive type). Casting may be necessary when an element is retrieved.

Array initializers

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

Array initializers: values must have the same type, in this case, int. Length of the array is the number of values in the list.

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.

```java
int[] z = new int[] {3, 2, 1};
String[] s = new String[2];
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.
Execution of the for-loop

The for-loop:
for (int i = 2; i <= 200; i++) {
    x = x + i*i;
}

loop counter: i
initialization: int i = 2;
loop condition: i <= 200;
repetend or body: { x = x + i; }

To execute the for-loop,
1. Execute initialization.
2. If loop condition false, terminate execution.
3. Execute repetend.
4. Execute increment, repeat from step 2.

Called a “flow chart”

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 = 2 values
2 .. 2 contains 2. It contains 2+1 - 2 = 1 values

The number of values in m..n is m + 1 - m.

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

The increment is executed after the repetend and before the next iteration.

What is the invariant?
1 2 3 ... k-1 k k+1 ... n