**CS1110 11 March 31 Testing/Debugging. Also, about A6**

Read chapter 14, pp. 385–401

Prelim 2 is Thursday eve, 14 April.

We will be contacting people about conflicts

Royal Flush is better than Full House

1

## Two-dimensional arrays

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| b | 5 | 4 | 7 | 3 |

b.length

one-dimensional array

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| d 0 | 5 | 4 | 7 | 3 |
| 1 | 4 | 8 | 9 | 7 |
| 2 | 5 | 1 | 2 | 3 |

rectangular array: 5 rows and 4 columns

Type of d is **int**[][] ("**int** array array", "an array of **int** arrays")

To declare variable d:

**int** d[][].

To create a new array and assign it to d:

d= **new int**[3][4];

To reference element at row r column c:

d[r][c]

number of rows

number of cols

2

## Testing: Read chapter 14.

**Bug**: Error in a program.

**Testing**: Process of analyzing, running program, looking for bugs.

**Test case**: A set of input values, together with the expected output.

**Debugging**: Process of finding a bug and removing it.

**Exceptions:** When an error occurs, like divide by 0, or s.charAt[i] when i = – 1, Java *throws an exception*. A lot — generally too much — information is provided.

Two ideas on test cases:
1. Black Box Testing: Develop test cases based on the spec.
2. White Box Testing: Look at the code; develop test cases so that each statement/expression is exercised in at least one test case.

3

**Exceptions:** When an error occurs, like divide by 0, or s.charAt[i] when i = – 1, Java *throws an exception*.

```
/** = String s truncated …. */
public static String truncate5(String s) {
    int b= 10 / 0;
    if (s.length() <= 5)
        return s;
    return s.substring(0,5);
}
```

Turn on line numbering in DrJava. Preferences / Display Options

important part

```
ArithmeticException: / by zero
 at A4Methods.truncate5(A4Methods.java:8)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(….java:39)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(….java:25)
 at java.lang.reflect.Method.invoke(Method.java:585)
```

call stack

4

## Debugging a program

When an error occurs, you have to play detective and find it. That process is called debugging. The place where the bug is may be far removed from the place where an error is revealed.

Strategy 0: Find a simplest possible test case that exhibits the error.

Strategy 1: put print statements, suitably annotated, at judiciously chosen places in the program.

Strategy 2: Use Java assert-statements at good places:
**assert** <boolean expression> ;

Strategy 3: Use the debugging feature of your IDE (Interactive Development Environment — yours is DrJava.

5

## Assert statement

Use it to program "defensively", and leave it in the program

Example: Use it to check preconditions:

```
/** = "This Virus is the predecessor of v".
      Precondition: v is not null */
public boolean isPredecessorOf(Virus v) {
   assert  v != null;
   …
}
```

6

## Debugging a program

When an error occurs, play detective and find it. Called debugging. The place where the bug is may be far removed from the place where an error is revealed.

```
public static HSV RGB2HSV(Color rgb) {
  …
  /**Set MAX, MIN to max and min of R, G, B */
  double MAX= 0; double MIN=  0;
  if (R>G && R>B)      {MAX= R; }
  if (G>B && G>R)     {MAX= G;}
  if (B>R && B>G)     {MAX= B;}
  if (R<G && R<B)     {MIN= R; }
  if (G<B && G<R)     {MIN= G; }
  if (B<R && B<G)     {MIN= B;}

  System.out.println("R " + R + ", G " + G +
                     ", B " + B + ", MAX " + MAX);
```

If you just output the numbers without naming them, you will have trouble.

7

## Debugging a program

When an error occurs, play detective and find it. Called debugging. The place where the bug is may be far removed from the place where an error is revealed.

```
public static HSV RGB2HSV(Color rgb) {
  …
  /**Set MAX, MIN to max and min of R, G, B */
  double MAX= 0; double MIN=  0;
  if (R>G && R>B)      {MAX= R; }
  if (G>B && G>R)     {MAX= G;}
  if (B>R && B>G)     {MAX= B;}
  if (R<G && R<B)     {MIN= R; }
  if (G<B && G<R)     {MIN= G; }
  if (B<R && B<G)     {MIN= B;}

  assert R <= MAX  &&  G <= MAX  &&  B <=  MAX;
  assert MIN <= R  &&  MIN <= G   &&  MIN <=  B;
```

These assert statements don't check completely that MAX is the max and MIN the min.

8

```
public static HSV RGB2HSV(Color rgb) {
  …
  if (R>G && R>B)      {MAX= R; }
  if (G>B && G>R)     {MAX= G;}
  if (B>R && B>G)     {MAX= B;}
  if (R<G && R<B)     {MIN= R; }
  if (G<B && G<R)     {MIN= G; }
  if (B<R && B<G)     {MIN= B;}
  System.out.println("R " + R + ", G " + G +
                     ", B " + B + ", MAX " + MAX);
```

call and output

> A4Methods.RGB2HSV(new java.awt.Color(255,255,128))
R 1.0, G 1.0, B 0.502, MAX 0.0

**Look! MAX is 0 and not 1!**

**if conditions should be >= , not >**

9

```
…
if (Hi ==0){
    R=(int)(v  * 255.0);
    G=(int)(t  * 255.0);
    B=(int)(p  * 255.0);
}
if (Hi==1){
    R=(int)(q  * 255.0);
    G=(int)(v  * 255.0);
    B=(int)(p  * 255.0);
}
…
System.out.println("In HSVtoRGB. R is " + R);
int r= (int)Math.round(R);
System.out.println("In HSVtoRGB. r is " + r);
```

Error in HSVtoRGB. Not rounding properly

Insert println statements.

10

## **Steganography**: hide a message in an image

P00 P01 P02 P03 P04 P05 P06 P07 P09 P10 P11 P12 P13

Message: Yea!

P00 P01 P02 P03 P04 P05 P06 P07 P09 P10 P11 P12 P13
Beginning markers Y e A ! End markers
(depends on design) (depends on design)

The time you spend on procedures hide and reveal will depend partially on your debugging technique. Write both hide and reveal and then see whether they work will waste your time.

It is best to write hide and test it thoroughly. THEN write reveal .

11

## Debugging hide (and reveal)

In ImageArray:
/** = pixel pix, in the form (red, green, blue) ... . */
**public static** String toString (**int** pix)

In the past, students have not debugged hide properly. Therefore, we make it easier to debug properly by requiring that you write (and use):

In ImageProcessor:
/** = a String containing the first n pixels of the current image, 5 to a line, with annotation (i.e. something at the beginning to say what the string contains). */
**public** String getPixels(**int** n)

12