CS100J 19 October 2006
About wrapper classes and class Vector
More on loops. Reading: Secs 7.1–7.3
Do the self-review exercises on pp. 235 and 242!!!

Quotes for the Day:

Instead of trying out computer programs on test cases until they are debugged, one should prove that they have the desired properties.
John McCarthy, 1961, A basis for a mathematical theory of computation.

Testing may show the presence of errors, but never their absence.
Dijkstra, Second NATO Conf. on Software Engineering, 1969.

A week of hard work on a program can save you 1/2 hour of thinking.
Paul Gries, CS, University of Toronto, 2005.

Wrapper classes and Java 1.5

An object of wrapper class Integer wraps (contains) a single **int** value. Allows you to view an **int** value as an object.

Integer d= **new** Integer(5);

d [a1]

a1: [5] Integer
MAX_VALUE
intValue()
equals() toString()

Primitive type	Wrapper class
byte	Byte
int	Int
short	Short
long	Long
double	Double
char	Character
boolean	Boolean

Two purposes of wrapper class:

1. An instance wraps a value of the primitive type.
2. The wrapper class provides several instance methods and static methods/fields to manipulate values of the primitive type.

Wrapper classes and Java 1.5
Java 1.5 makes it easier to deal with the wrapper classes

An object of wrapper class Integer wraps (contains) a single **int** value. Allows you to view an **int** value as an object.

Integer d= **new** Integer(5);

d [a1]

a1: [5] Integer
MAX_VALUE
intValue()
equals() toString()

Java 1.4
int i= d; illegal
Have to write:
int i= d.intValue();

Integer f= 5; illegal
Have to write:
Integer f= **new** Integer(5);

Java 1.5
int i= d; legal
It does the equivalent of:
int i= d.intValue();

Integer f= 5; legal
It does the equivalent of:
Integer f= **new** Integer(5);

About class Vector (in lab today)

0 1 2 3 4 5 6 7 8
v [X Y Z X A C Z Z Z] This is a Vector of Characters

Java 1.4:
Vector v= **new** Vector();

An object of class Vector contains a list of objects, all of the are Objects

v.add(ob) add object obj to v
v.get(0) = first object in v
v.get(1) = second object in v
v.size() = no. of objects in v

If you know v.get(0) is of class Character, can do

(Character) v.get(0)

Java 1.5:
Vector<Character> v=
new Vector<Character>();

The objects in v have to be of type Character.

v.add(ob) add object obj to v
v.get(0) = first object in v
v.get(1) = second object in v
v.size() = no. of objects in v

If you know v.get(0) is of class Character, can do

Character c= v.get(0); this works

Understanding assertions

0 1 2 3 4 5 6 7 8
v [X Y Z X A C Z Z Z] This is a Vector of Characters

This is an assertion about v and k. It is **true** because chars of v[0..3] are greater than 'C' and chars of v[6..8] are 'Z's.

0 3 k 8
v [≥ C | ? | all Z's] k [6]

0 3 k 8
v [≥ C | ? | all Z's] k [5]

0 k 8
v [≥ C | all Z's] k [6]

0 k 8
v [≥ W | ? | ? | all Z's] k [4]

Indicate whether each of these 3 assertions is true or false.

The while loop

x= 0;
x= x + 2*2;
x= x + 3*3;
x= x + 4*4;

```
x= 0;
int k= 2;
while ( k != 5) {
    x= x + k*k;
    k= k+1;
}
```

To execute the while loop:

(1) Evaluate condition k != 5;
if false, stop execution of loop.

(2) Execute the repetend.

(3) Repeat again from step (1).

Repetend: the thing to be repeated. The block:

```
{
 …
}
```

Develop loop to store in x the sum of 1..100.

We'll keep this definition of x and k true:
x = sum of 1..k–1

1. How should the loop start? Make range 1..k–1 empty: **k= 1; x= 0;**

2. When can loop stop? What condition lets us know that x has result? When **k == 101**

3. How can repetend make progress toward termination? k= k+1;

4. How do we keep def of x, h, k true? x= x + k;

Four loopy questions

```
k= 1; x= 0;
// invariant: x = sum of 1..(k–1)
while ( k != 101) {
    x= x + k;
    k= k + 1;
}
// { x = sum of 1..100 }
```

7

Develop loop to store in x the sum of 1..100.

This time, we'll keep this definition of x and k true:
x = sum of h..100

1. How should the loop start? Make range h..100 empty: **h= 101; x= 0;**

2. When can loop stop? What condition lets us know that x has result? When **h == 1**

3. How can repetend make progress toward termination? h= h – 1;

4. How do we keep def of x, h, k true? x= x + (h – 1);

Four loopy questions

```
h= 101; x= 0;
// invariant: x = sum of h..100
while ( h != 1) {
    x= x + (h – 1);
    h= h – 1;
}
// { x = sum of 1..100 }
```

8

Roach infestation!

```
/** = number of weeks it takes roaches to fill the apartment --see p 244 of text*/
public static int roaches() {
    double roachVol= .001;    // Space one roach takes
    double aptVol= 20*20*8;   // Apartment volume
    double growthRate= 1.25;  // Population growth rate per week

    int w= 0;         // number of weeks
    int pop= 100;     // roach population after w weeks

    // inv: pop = roach population after w weeks   AND
    //      before week w, volume of the roaches < aptVol
    while (aptVol > pop * roachVol ) {
        pop= (int) (pop * growthRate);
        w= w + 1;
    }
    return w;
}
```

9

Logarithmic algorithm to calculate b**c, for c >= 0

```
/** = b**c, given c ≥ 0 */
public static int exp(int b, int c) {
    if (c == 0) return 1;
    if (c%2 = 0) return exp(b*b, c/2);
    return b * exp(b, c–1);
}
```

Rest on identities:

b**0 = 1

b**c = b * b**(c-1)

for even c, b**c = (b*b)**(c/2)

3*3 * 3*3 * 3*3 * 3*3 = 3**8

(3*3)*(3*3)*(3*3)*(3*3) = 9**4

Algorithm processes binary representation of c

Suppose c is 14 (1110 in binary)

1. Test if c is even: test if last bit is 0

2. To compute c/2 in binary, just delete the last bit.

Algorithm processes each bit of c at most twice.

So if c is 2**15 = 32768, algorithm has at most 2*15 = 30 recursive calls!

Algorithm is *logarithmic in c*, since time is proportional to log c

10

Iterative version of logarithmic algorithm to calculate b**c,
for c >= 0 (i.e. b multiplied by itself c times)

```
/** set z to b**c, given c ≥ 0 */
int x= b; int y= c; int z= 1;
// invariant: z * x**y = b**c and 0 ≤ y ≤ c
while (y != 0) {
    if (y % 2 == 0)
         { x= x * x; y= y/2; }
    else { z= z * x; y= y – 1; }
}
// { z = b**c }
```

Rest on identities:

b**0 = 1

b**c = b * b**(c-1)

for even c, b**c = (b*b)**(c/2)

3*3 * 3*3 * 3*3 * 3*3 = 3**8

(3*3)*(3*3)*(3*3)*(3*3) = 9**4

Algorithm is *logarithmic in c*, since time is proportional to log c

11

Calculate quotient and remainder when dividing x by y

x/y = q + r/y   **21/4= 4 + 3/4**

Property: x = q * y + r and 0 ≤ r < y

```
/** Set q to and r to remainder.
    Note: x >= 0 and y > 0 */
int q= 0; int r= x;
// invariant: x = q * y + r   and 0 ≤ r
while (r >= y) {
    r= r – y;
    q= q + 1;
}
// { x = q * y + r   and  0 ≤ r < y }
```

12