Packages & Bits

Discussion 06

Announcements

- Lab 5 due Tuesday 2/22
- Enigma Checkpoint due Friday 2/25
- HW 4 due Tuesday 3/1

Review

Access Modifiers

Private is the tightest level of privacy - variables and functions with this modifier can only be accessed by the same class.

Package-Private is the default level of privacy - variables and functions with this modifier can be accessed by classes within the same package but not outside classes, including subclasses.

Protected is similar to package private except through subclasses.

Public is the loosest level of privacy - variables and functions with this modifier can be accessed by all other classes.

Bitwise Operations

Mask (And)

```
  01101011
& 10100101
  00100001
```

Set (Or)

```
  01101011
| 10100101
  11101111
```

Flip (Xor)

```
  01101011
^ 10100101
  11001110
```

Flip All (Neg)

```
~ 10100101
  01011010
```

Shift Left

```
   11101011
<<        3
   01011000
```

Shift Logical Right

```
   11101011
>>>       3
   00011101
```

Shift Arithmetic Right

```
  11101011
>>       3
  11111101
```

CS 61B Spring 2022 — Packages and Bits

Exam Prep Discussion 6: February 21, 2022

1 Packages Have Arrived

In the following classes, cross out the lines that will result in an error (either during compilation or execution). Next to each crossed-out line write a replacement for the line that correctly carries out the evident intent of the erroneous line.

Each replacement must be a single statement. Change as few lines as possible.

After your corrections, what is printed from running **java P2.C5**?

```
1  package P1;
2  class C1 {
3      private int a = 1;
4      protected int b = 2;
5      int c = 3;
6
7      public static int d() {
8          return 13;
9      }
10     public void setA(int v) { a = v; }
11     public void setB(int v) { b = v; }
12     public void setC(int v) { c = v; }
13     public int getA() { return a; }
14     public int getB() { return b; }
15     public int getC() { return c; }
16
17     public String toString() {
18         return a + " " + getB() + " " + getC() + " " + d();
19     }
20 }
21 ------------------------------------------------
22
23 package P1;
24 class C2 extends C1 {
25     public C2() {}
26     public C2(int a, int b, int c) {
27         this.a = a;
28         this.b = b;
29         this.c = c;
30     }
31     public static int d() {
32         return 14;
33     }
```

Write output here: (written in P2.C5)

_____

_____

_____

public ↑

default (package-private) ↑

public ↑

More Restrictive ↓

Public - anything can access
Protected - same package or any subclass
Default - same package
Private - only the class itself

First make sure code compiles.
Then simulate runtime and evaluate.

this.a = a; → setA(a);

```
34      public C1 gen() {
35          return new C3();
36      }
37  }
38  ---------------------------------------------
39
40  package P1;
41  class C3 extends C2 {
42      private int a = 15;
43      public String toString() {
44          return a + " " + getB() + " " + getC() + " " + d();
45      }
46  }
47  ---------------------------------------------
48
49  package P2;
50  class C4 extends C2 {
51      public int getB() {
52          return 2 * b;
53      }
54      public C4(int a, int b, int c) {
55          this.a = a;
56          this.b = b;
57          this.c = c;
58      }
59      public C4(int v) {
60          this.a = this.b = this.c = v;
61      }
62  }
63  ---------------------------------------------
64
65  package P2;
66  class C5 {
67      public static void main(String... args) {
68        P1.C1 x = new P1.C1();
69        P1.C2 y = new C4(20, 30, 40);
70        P1.C3 z = y.gen();
71              (P1.C3)
72        System.out.println(x);
73        System.out.println((P1.C2) y);
74        System.out.println(z);
75      }
76  }
```

public

P1.

Overrides

this.a = a; setA(a);

this.c = c; setC(c);

this.a = this.b = this.c = v; super(v, v, v);

Casting

from class C1

1 2 3 13

20 60 40 13

getB is overriden, use toString of P1.C2

15 2 3 14

different d method being used

2 Bit Operations

In the following questions, use bit manipulation operations to achieve the intended functionality and fill out the function details -

(a) Implement a function `isPalindrome` which checks if the binary representation of a given number is palindrome. The function returns true if and only if the binary representation of `num` is a palindrome. Assume `num` is 32 bits.

For example, the function should return true for `isPalindrome(0xDEADDAED)` since binary representation of 9 is `1001` which is a palindrome.

```
1  /**
2  * Returns true if binary representation of num is a palindrome
3  */
4  public static boolean isPalindrome(int num) {
5
6      int reverse = 0;
7
8      int k = num;
9
10     while (k > 0) {                          last bit of k
11                                                  ↓
12         reverse = (reverse << 1) | (k & 1);
13
14         k = k >> 1;
15                  ↑ "increments"
16     }
17
18     return num == reverse;
19 }
```

(b) Implement a function `swap` which for a given integer, swaps two bits at given positions. The function returns the resulting integer after bit swap operation.

For example, when the function is called with inputs `swap(31, 3, 7)`, it should reverse the 3rd and 7th bits from the right and return 91 since 31 (00011111) would become 91 (01011011).

```
/**
* Function to swap bits at position a and b (from right) in integer num
*/
public static int swap(int num, int a, int b) {
    int bit_a = (num >> a-1) & 1;

    int bit_b = (num >> b-1) & 1;

    if (bit_a != bit_b) {

        num ^= (1 << a-1)                    ← flips bits

        num ^= (1 << b-1)

    }

    ____________________________________________

    return num;
}
```

3 Bits Runtime

Determine the best and worst case runtime of `tricky`.

```
public void tricky(int n) {
   if (n > 0) {
       tricky(n & (n - 1));
   }
}
```

Best Case: $\Theta(\ 1\ )$, Worst Case: $\Theta(\log N)$