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;                                          Write output here:
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     }
```

```
    public C1 gen() {
        return new C3();
    }
}
```

```
package P1;
class C3 extends C2 {
    private int a = 15;
    public String toString() {
        return a + " " + getB() + " " + getC() + " " + d();
    }
}
```

```
package P2;
class C4 extends C2 {
    public int getB() {
        return 2 * b;
    }
    public C4(int a, int b, int c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public C4(int v) {
        this.a = this.b = this.c = v;
    }
}
```

```
package P2;
class C5 {
    public static void main(String... args) {
        C1 x = new C1();
        C2 y = new C4(20, 30, 40);
        C3 z = y.gen();

        System.out.println(x);
        System.out.println((P1.C2) y);
        System.out.println(z);
    }
}
```

Solution:

Here is a video walkthrough of the solution.

```
package P1;

public class C1 {

    private int a = 1;
    protected int b = 2;
    int c = 3;

    public static int d() {
        return 13;
    }
    public void setA(int v) { a = v; }
    public void setB(int v) { b = v; }
    public void setC(int v) { c = v; }
    public int getA() { return a; }
    public int getB() { return b; }
    public int getC() { return c; }

    public String toString() {
        return a + " " + getB() + " " + getC() + " " + d();
    }
}
```

Write output here:

____ 1 2 3 13 _____

____ 20 60 40 13 ______

____ 15 2 3 14 ________

```
package P1;

public class C2 extends C1 {

    public C2() {}
    public C2(int a, int b, int c) {

        setA(a);

        this.b = b;
        this.c = c;
    }
    public static int d() {
        return 14;
    }
    public C1 gen() {
        return new C3();
    }
}
```

```
package P1;

public class C3 extends C2 {

    private int a = 15;
    public String toString() {
```

```
        return a + " " + getB() + " " + getC() + " " + d();
    }
}
```

```
package P2;

class C4 extends P1.C2 {

    public int getB() {
        return 2 * b;
    }
    public C4(int a, int b, int c) {

        setA(a);

        this.b = b;

        setC(c);

    }
    public C4(int v) {

        super(v,v,v);

    }
}
```

```
package P2;
class C5 {
    public static void main(String... args) {

        P1.C1 x = new P1.C1();

        P1.C2 y = new C4(20, 30, 40);

        P1.C3 z = (P1.C3) y.gen();

        System.out.println(x);
        System.out.println((P1.C2) y);
        System.out.println(z);
    }
}
```

Fixes:
The following lines need to be fixed:
Line 2: In order to access `C1` in another package `P2`, it needs to be a `public class`.
Line 23: Similar logic to line 2–in order to access `C2` in another package, it needs to be a `public class`.
Line 27: `a` is a private variable, so subclasses like `C2` cannot access it directly–instead, it must use the `setA` method.
Line 41: Similar to line 2 and 23.

Line 50: Since we have not imported `P1`, we must use preface the class we want to use by its full package name.

Line 55: Similar to Line 27, `a` is a private variable.

Line 57: A variable with no declared access modifier is package private, which means it can only be accessed within the same package. `C4` is in package `P2`, so it must use the public method `setC` instead of directly accessing `c`.

Line 60: Again, `C4` cannot directly access `a` or `c`, but it can call its parent's constructor with `super`, which achieves the desired effect.

Line 68, 69: Similar to Line 50; without importing `P1`, the full package name must be used.

Line 70: `y` has static type `C2`, and `C2.gen` has return type `C1`. However, we know that the method actually returns an object of type `C3`, so casting allows the assignment to compile.

Print output:

Line 72: `C1`'s `toString` method simply prints the `a, b, c`, and `d()` values.

Line 73: `C4` has `a, b, c` as `20, 30, 40` respectively. It inherits the `toString` method from `C1`. However, its `getB` method overrides the `getB` method in `C1`, so `getB()` will return 60. Note that the `d()` method is static, so the method that is run is decided at compile time (there is no overriding for static methods). Thus, at compile time, the compiler finds `getString()` inside of `C1` and also uses `C1`'s `d()` method. The final values printed are `20, 60, 40, 13`.

Line 74: `y.gen()` calls the no-argument `C3` constructor. Note that `C3` has its own `private a` with a value of 15. However, it still inherits `b` and `c` from `C1`, and its `d()` method from `C2`. This gives the final output `15 2 3 14`.

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.

```
/**
* Returns true if binary representation of num is a palindrome
*/
public static boolean isPalindrome(int num) {

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________
}
```

Solution:

```
/**
* Returns true if binary representation of num is a palindrome
*/
public static boolean isPalindrome(int num) {
    // stores reverse of binary representation of num
    int reverse = 0;

    // do till all bits of num are processed
    int k = num;
    while (k > 0) {
        // add rightmost bit to reverse
        reverse = (reverse << 1) | (k & 1);
        k = k >> 1;             // drop last bit
```

```
14      }
15      return num == reverse;
16  }
```

Explanation: The main idea is to reverse the bits of `num`; it is a palindrome if and only if it is equal to its reverse. To do this, we initialize `reverse` to all zeros. Inside the loop:

1. Shift `reverse` to "vacate" its last bit.
   `rrr << 1 -> rrr0`
2. Get the last bit of `k`.
   `kkkk & 0001 -> 000k`
3. `or` the numbers together to get the combined bits.
   `rrr0 | 000k -> rrrk`
4. Remove the bit of `k` we just used.

(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) {
    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    return num;
}
```

Solution:

```
/**
* 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 p = a-1;
    int q = b-1;

    int bit_a = (num >> p) & 1;
    int bit_b = (num >> q) & 1;

    if (bit_a != bit_b) {       // if the bits are different
        num ^= (1 << p);
        num ^= (1 << q);
    }
    return num;
}
```

Explanation: To get the kth bit from the right in a number, we can shift the number right by `k - 1` bits, then perform an `&` with 1. For a visualization, suppose we are trying to get the third bit from the right for $b_4b_3b_2b_1$. First, we right shift by 2 to get $00b_4b_3$. $00b_4b_3$ & 0001 gives $000b_3$ as desired. This is the operation performed in line 8 and 9.
We only need to swap if the two bits are different. If the bits are different, this problem reduces to flipping the bits at position `a` and `b`. To flip a bit at position `k`, we simply `xor` it with 1 ($1 \oplus 1 = 0, 0 \oplus 1 = 1$). This corresponds to lines 12 and 13.

3 Bits Runtime

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

```
1  public void tricky(int n) {
2     if (n > 0) {
3         tricky(n & (n - 1));
4     }
5  }
```

Best Case: $\Theta($ $)$, Worst Case: $\Theta($ $)$

Solution:
Best Case: $\Theta(1)$, Worst Case: $\Theta(logN)$
Explanation: The main idea is that this function zeros out a `1` in `n` each time. If `n` starts off as some power of 2, it only has one `1` and finishes in constant time. If `n` is all ones, it takes `log N` recursive calls to finish (there are `log N` bits in N).
There are two main cases for `n`. First, if `n` is odd, `n - 1` has a `0` in the last bit, so the last bit of `n` will be zeroed out. If `n` is even so its last bits are something like `10 ... 0`, then the last bits of `n - 1` will be `01 ... 1`. and-ing these together zeros out the first nonzero bit from the right.