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`?

```java
package P1; Write output here:
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();
    }
}
package P1;
class C2 extends C1 {
    public C2() {}
    public C2(int a, int b, int c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public static int d() {
        return 14;
    }
```
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();
    }
}

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();
    }
}
    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.

```java
/**
 * 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
    }
}
```

Solution:

```java
/**
 * 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
    }
}
```
```cpp
return num == reverse;
```

**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.
   \[ rrr << 1 \rightarrow rrr0 \]

2. Get the last bit of `k`.
   \[ kkk & 0001 \rightarrow 000k \]

3. `or` the numbers together to get the combined bits.
   \[ rrr0 | 000k \rightarrow rrrk \]

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).

```java
/**
* 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;
}
```

Solution:
**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_1b_2b_3b_4. First, we right shift by 2 to get 00b_3b_4. 00b_3b_4 & 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 ⊕ 1 = 0, 0 ⊕ 1 = 1). This corresponds to lines 12 and 13.

### 3 Bits Runtime

Determine the best and worst case runtime of tricky.

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

**Solution:**

Best Case: Θ(1), Worst Case: Θ(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.