Practice Final Exam Answers

Section 1

1. C is false. Representation invariants are usually specified in comments which are ignored by the compiler.
2. D cannot be guaranteed. The compute method may return an int value.
3. E. No matter how many times you multiply 0 by 2, it is still 0 and will always be less than 10.
4. D (assuming x <= 11).
5. C.
6. B. If the array does not have an i\textsuperscript{th} element, an ArrayIndexOutOfBoundsException may be thrown.
7. B and C. If either B or C is true, the isFull() expression will be skipped by Java.
8. B.
9. D is false. The method is static, there is no ‘this’.
10. False. Static methods may be private.
11. C. Converting a double to an int always truncates the number (the fractional portion is dropped).
12. D if false. The compiler may view the object as if it were a different type, but the content of the object is not changed.
13. C is false. Subclass methods override superclass methods.
14. B. Both variables now contain the same reference.
15. A would be illegal. While it is true that Alpha objects are Gamma objects, Gamma objects are not Alpha objects and cannot be stored in Alpha variables.
16. D would throw the exception. Same logic as in 15, Beta objects are not Alpha objects.
17. A.
18. D.
19. C.
20. B.
21. C.
22. D.
23. A.
24. True.
25. False.
26. True.
27. False. Abstract Data Types do not require any particular implementation details.
Section 2

Method #1:

```java
public static int sum (ArrayList<Integer> list)
{
    int largest;
    int smallest;

    largest = list.get(0);
    smallest = list.get(0);

    for (int i : list)
    {
        if (i < smallest)
            smallest = i;
        if (i > largest)
            largest = i;
    }

    return smallest + largest;
}
```

Method #2:

```java
public static void reverse(Object[] a)
{
    for (int i = 0; i < list.length / 2; i++)
    {
        int otherPos = (list.length - i) - 1;

        Object temp = a[i];
        a[i] = a[otherPos];
        a[otherPos] = temp;
    }
}
Section 3

One possibility:

```java
public double calculateSalary (int years, double base)
{
    if (years <= 5)
        return base;

    int bonusYears = years - 5;
    double tenPercentOfBase = base * 0.10;
    double salary = base + tenPercentOfBase * bonusYears;

    if (salary > base * 2)
        salary = base * 2;

    return salary;
}
```

Another possibility:

```java
public double calculateSalary (int years, double base)
{
    if (years <= 5)
        return base;

    if (years >= 15)
        return base * 2;

    return base + (base * 0.10) * (years - 5);
}
Section 4

5-1: private String description;
5-2: private double price;
5-3: private int quantity;

5-4: this.description = description;
5-5: this.price = price;
5-6: this.quantity = quantity;

Alternate for 5-5:

5-5: setPrice (price);

5-7: return description;

5-8: return price * quantity;

5-9: quantity = quantity + additionalItemCount;

5-10: if (isInStock())
       quantity--;

5-11: item1 = null;
5-12: item2 = null;
5-13: item3 = null;

5-14: int count = 0;
    if (item1 != null)
      count++;
    if (item2 != null)
      count++;
    if (item3 != null)
      count++;
    return count;

5-15: double value = 0.0;
    if (item1 != null)
      value = value + item1.getTotalValue();
    if (item2 != null)
      value = value + item2.getTotalValue();
    if (item3 != null)
      value = value + item3.getTotalValue();
    return value;
Section 5

public class MobileBob implements Mobile
{
    // I skipped instance variable and constructors, these
    // are listed on the exam. All you would need to do
    // is write the methods required from the Mobile interface.

    public int totalWeight ()
    {
        return mass;
    }

    public boolean isBalanced ()
    {
        return true;
    }

    public doubleAllMasses ()
    {
        mass = mass * 2;
    }
}

public class MobileRod implements Mobile
{
    // I skipped instance variable and constructors, these
    // are listed on the exam. All you would need to do
    // is write the methods required from the Mobile interface.

    public int totalWeight ()
    {
        return left.totalWeight() + right.totalWeight();
    }

    public boolean isBalanced ()
    {
        return left.totalWeight() == right.totalWeight();
    }

    public doubleAllMasses ()
    {
        left.doubleAllMasses();
        right.doubleAllMasses();
    }
}
public void swap (E a, E b) {
    // Find the node that contains element a.
    Node currentA = head;
    Node previousA = null;

    while (currentA != null && !currentA.containsData(a)) {
        previousA = currentA;
        currentA = currentA.getNext();
    }

    // Find the node that contains element b.
    Node currentB = head;
    Node previousB = null;

    while (currentB != null && !currentB.containsData(b)) {
        previousB = currentB;
        currentB = currentB.getNext();
    }

    // If either node is null, bail out - nothing to do.
    // If the nodes are the same, bail out.
    if (currentA == null || currentB == null
        || currentA == currentB)
        return;

    // Create two new nodes for a and b.
    Node newA = new Node(a);
    Node newB = new Node(b);

    // Replace existing nodes with their new replacements.
    newA.setNext(currentB.getNext());
    if (previousB == null)
        head = newA;
    else
        previousB.setNext(newA);
newB.setNext(currentA.getNext());
if (previousA == null)
    head = newB;
else
    previousA.setNext(newB);

// Adjust tail as needed.

if (currentA == tail)
tail = newB;

if (currentB == tail)
tail = newA;