

Quiz Solutions

Note: these answers are based on the ‘B’ version of the quiz. If you wrote the ‘A’ version, the same questions appear, but in a different order

2. **TRUE** (a) or **FALSE** (b): A `File` object can be used to programmatically change a file's name on a USB drive
3. **TRUE** (a) or **FALSE** (b): `Scanner` cannot be used for file IO
4. **TRUE** (a) or **FALSE** (b): when a new object is instantiated, it's superclass no-arg constructor is run first
5. Which one of the following is *NOT* a typical use for a static member?
 - (a) It can serve as a common location that all objects can reference
 - (b) It can be used when an operation is not instance-dependent
 - (c) It can be used to reduce space requirements in systems with restricted RAM space
 - (d)** It can be used to access instance members of the same class
 - (e) none of the above

Sample Quiz Solutions

6. When a constructor has the same name but a different signature, it is said to be
(a) overloaded (b) overridden (c) chained (d) static
(e) none of the above
7. Which one of the following statements is **NOT TRUE** of the *enhanced for* loop
(a) Unlike the regular *for* loop, it uses the colon (:), not the semicolon (;) in its format
(b) You can only count up with an *enhanced for*; you cannot count down.
(c) You can get data *out* of an array using the *enhanced for* but you cannot put data into an array (unless you use an index)
(d) You do not need to go through every element in the array first, you can break out of the *enhanced for* prematurely, using the `break` command
(e) none of the above
8. When the following code fragment is executed

```
short num = (byte) 42;  
float ratio = num/10.0f;
```


the result of this calculation will be
(a) 4 (b) 4.0 (c) 4.2 (d) a compile-time error (e) none of the above

9. For a class `ObjectCounter` having the following method

```
public static void setCounter(int newCtr) {...}
```

whenever an integer value is passed as a parameter to `setCounter` (for example, the value 5 in `ObjectCounter.setCounter(5)`), the method's parameter, `int newCtr`, acts as a

- (a) declaration (b) definition (c) assignment (d) static variable (e) none of the above
10. In question 9 above, `setCounter()` may be described as a/n
- (a) static property (b) static method (c) instance method (d) iterator
(e) none of the above
11. Which one of the following is *NOT* a method of the `Object` class
- (a) `clone` (b) `equals` (c) `toString` (d) `finalize` (e) none of the above
12. **TRUE** (a) or **FALSE** (b): A constructor will always have the same name as the name of the file (minus the `.java` extension) in which it is loaded
13. If a class is declared `public` then its visibility is limited to
- (a) the class (b) the package (c) subclasses (d) the entire project
(e) none of the above

14. **TRUE** (a) or **FALSE** (b): UML allows the user to omit information when it is not essential to understanding the relationship between classes.

15. In debug mode, to *resume* execution, you would use:







- (a) (or F8) b) (or F5) c) (or F6) d) (or Ctl-F11) e) none of the above

16. **TRUE** (a) or **FALSE** (b): Provided their signatures are identical, it is not necessary to use `@Override` to ensure that a subclass method successfully overrides a superclass method.

17. If the class in question 9 was written in a UML class diagram, it will appear as:

- (a) `+setCounter(newCtr: int): void`
(b) `-setCounter(newCtr: int): void`
(c) `+setCounter(newCtr: int): void`
(d) `+setCounter(): void`
(e) none of the above

18. **TRUE** (a) or **FALSE** (b): Superclass methods can always be invoked using `super.methodName()`

19. Which one of the following is true of an abstract class:
- (a) It cannot contain a static method
 - (b) All its methods become abstract
 - (c) It can be instantiated into an object
 - (d) It does not have access to the `Object` class's methods
 - ☒ (e) None of the above
20. Which one of the following symbols is used to indicate that one class has another class (i.e. a 'has a' relationship exists between two classes):
- (a)  (b) ☒  (c)  (d)  (e) none of the above
21. **TRUE** ☒ (a) or **FALSE** (b): Once a `final` property has a value assigned, it cannot be changed
22. Which symbol is used to indicate an annotation?
- (a) - (b) + (c) # ☒ (d) @ (e) none of the above
23. In a UML diagram, constants are identified by being:
- a) underlined b) *italicized* c) **bold-face** ☒ (d) CAPITALIZED e) none of the above

Part B: Terminology – worth 1 mark each

FILL IN THE SINGLE BEST ANSWER—ONE WORD OR SYMBOL IN THE EACH SPACE PROVIDED BELOW.

USE ONE WORD ONLY IN EACH SPACE; DO NOT USE ACRONYMS

24. In Unit/ JUnit testing, the test classes should be stored in a separate package from the one used by the regular executable classes of the program.
25. The keyword `extends` is used in the Java language to indicate that a class inherits the public members of its parent class
26. Rather than attempting to change the name of a class and all its dependencies (like the constructors) directly, you should refactor it by right-clicking the class name and selecting this item from the Eclipse menu.
27. In Java, each subclass may only have *one* parent class. This is known as single inheritance (2 words)
28. A parent class is to a child class as a base class is to a derived class

Part C: Written Answers – worth 8 marks total

In *Fruit.java*:

```
public abstract class Fruit {  
    private String name, colour;  
    private float weight;  
    1 public Fruit(String name, float weight){  
        2 super(name, weight, "");  
    }  
    private Fruit(String name, float weight,  
        String colour){  
        setName(name);  
        setWeight(weight);  
        setColour(colour);  
    }  
    3 public Fruit(String colour, float weight){  
        this("", 1.0, colour);  
    }  
    4 public double getWeight(){return weight;}  
    5 public void setWeight(float weight){  
        this.weight = weight;    }  
    public String getName(){return name;}  
    public void setName(String name){  
        this.name = name;    }  
    public String getColour(){return colour;}  
    public void setColour(String colour){  
        this.colour = colour;  
    }  
}
```

1. Missing no-arg constructor in superclass
2. use of super in constructor refers to Object, which does not have String, float, String constructor
3. Two constructors have same signature, i.e. String, float
4. attempt to pass a double value (1.0) as second parameter, when float expected; need to downcast second parameter, or use 1.0f as the default value passed to the second constructor
5. Getter returns a double, but weight is a float

In *Orange.java*:

6

```
public class Orange extend Fruit {
    public Orange orangeTree[];
    7 private final int numberOfOranges = 3;
    private String colour;

    public Orange(String name, float size){
        8 super(weight, name, size);}
    9 @Override
    10 public String toString(){
        11 System.out.println(this.getName()+" has weight
            "+this.getWeight()+" gm");
    }

    public String getColour(){return colour;}
    public void setColour(){this.colour = colour;}

    public static void main(String[] args){
        orangeTree = new Orange[numberOfOranges];
        System.out.println("Inputing oranges into array
            now");
        for (int i = 0; i < numberOfOranges;)
            orangeTree[i] = new Orange("Oranges No " +
                ++i, (500 + 50*i)); 12 13
        System.out.println("Total times through loop is "
            + i); 14
        System.out.println("\nOutputing oranges from
            array");
        for (Orange thisOrange: orangeTree)
            System.out.println(thisOrange.toString());
    }
}
```

6. extends, not extend
7. missing static in declaration of OrangeTree[] and numberOfOranges. main knows nothing about these values, since they are part of an instance of the class, and not the class itself.
8. No matching constructor in superclass
9. weight is inaccessible in the subclass, since it is private in the superclass (but could use getWeight() here instead, since getWeight() is public in the superclass)
10. @Override, not @Override
11. toString() must return string, not print it
12. new Orange () attempts to pass (String name, int weight) to constructor that takes a (String name, float size) instead. The int gets automatically upcast to a float with no problems; but then you get a weight loaded as the size.
13. pre-increment of i means that each weight will be 50 more than output shows
14. i is out of scope outside the for loop

Suggested ‘errors’, which are not real errors, in Fruit.java

| Suggested Error | Why it isn’t an error |
|--|--|
| Missing package | Doesn’t need one, since the classes could each be in the <code>default</code> package |
| Needs import <code>lang.util.Scanner</code> | Code doesn’t use <code>Scanner</code> ; no need to import it |
| Fruit class can’t be <code>abstract</code> | Of course it can; Fruit is a typical abstract class. Just don’t try to instantiate it |
| <code>name</code> and <code>colour</code> need to be declared on separate lines | Nope, this works just fine as is (this single-line type of declaration has been used in the examples in the lecture notes and in the lab) |
| Can’t use <code>super</code> in Fruit constructor, since Fruit is the superclass | Fruit is a subclass of <code>Object</code> , hence you <i>can</i> use <code>super</code> . The real problem is that <code>Object</code> lacks a constructor that takes as parameters a <code>String</code> , <code>float</code> , and <code>String</code> . But nothing prevents you from calling <code>super</code> in a superclass, provided of course that that class isn’t the <code>Object</code> class, which alone of all classes has no superclass |
| Didn’t pass <code>colour</code> or <code>name</code> in the first and third constructor | The first and third constructors are chained to the second constructor, which takes 3 parameters. The defaults passed to it are just empty strings, <code>""</code> . |
| The second constructor should be <code>public</code> , not <code>private</code> | As long as you don’t try to call it directly from a subclass there’s no problem. Again, you may not like it, but it’s <i>not an error</i> , since private constructors are OK |
| The sequence of parameters in any <code>this()</code> statement must match that of the constructor in which it is found. | No. Just because the third constructor passes a (<code>String colour</code> , <code>float name</code>) in its parameter list, that certainly doesn’t mean you need to chain items in <code>this()</code> in exactly the same order. |

Suggested 'errors', which are not real errors, in Orange.java

| Suggested Error | Why it isn't an error |
|--|---|
| <code>public Orange orangeTree[];</code> is not a proper declaration | This is a proper <i>declaration</i> for an array of objects (it's what appears on the LHS of the '=' sign). The actual <i>assignment</i> is in the code below. Location of the brackets can go on either side of identifier. Again, see the hybrids on this. |
| <code>numberOfOranges</code> should be capitalized | First, this isn't a genuine constant like <code>PI</code> ; its value is arbitrary. Second, capitalization is not a requirement of the JVM, hence this won't generate any errors |
| <code>name</code> is not declared in the <code>Fruit</code> class, therefore it can't be passed as a parameter | <code>name</code> is just an identifier used to pass a <code>String</code> , so the fact that there is no <code>private name</code> declared in <code>Fruit</code> is irrelevant. Nothing prevents the user from passing a default name using <code>new Orange("Oranges No " + ++i, (500 + 50*i));</code> |
| There is no <code>toString()</code> in the superclass to override | There's a <code>toString()</code> in <code>Object</code> , which you inherit via <code>Fruit</code> . So yes, there is a <code>toString()</code> method in the subclass to override. |
| Can't redeclare <code>getColour()</code> and <code>setColour()</code> in subclass | Since the signatures are the same, they automatically override the superclass versions of <code>getColour()</code> and <code>setColour()</code> . Unusual, but again, nothing wrong with it. |
| <code>for</code> loop is wrong: failure to increment <code>i</code> , or extra ';' at end of loop header | Nothing wrong with the <code>for</code> loop, as indicated in hybrid module on loops. The loop control variable, <code>i</code> , is incremented as each new <code>Orange</code> is instantiated, so the loop does NOT go on forever |
| <code>for</code> loop needs parentheses { } | Not if the loop only contains a single line; it works just fine |
| Should be <code>this.Orange</code> | No; at best this would be redundant, since <i>this is the</i> <code>Orange</code> . <code>thisOrange</code> is the identifier that holds each element of the array of <code>Oranges</code> . Again, use of the <i>enhanced for</i> was covered in the hybrid on loops. |