Homework 6

Q1)(C)

Java has four access modifiers:

- `public` (least restrictive)
- `protected`
- `default`
- `private` (most restrictive)

The members of a class defined using the protected access modifier are accessible to

- Classes and interfaces defined in the same package
- All derived classes, even if they're defined in separate packages

The members of a class defined without using any explicit access modifier are defined with package accessibility (also called default accessibility). The members with package access are only accessible to classes and interfaces defined in the same package. The Default access is also referred to as package-private.

Q2)(B)

It's common to define multiple constructors in a class and reuse their functionality across constructors. Unlike overloaded methods, which can be invoked using the name of a method, overloaded constructors are invoked by using the keyword `this`, an implicit reference that's accessible to all objects that refer to an object itself.

The reference variable super can be used to refer to the constructors of the base class in a derived class. If present, a call to a superclass's constructor must be the first statement in a derived class's constructor. Otherwise, a call to super(); (the no-argument constructor) is inserted automatically by the compiler.

Q3)(D)

The code does not compile. If both of the if-then statement's conditions returns false, function does not return a boolean value while it is declared as to return a boolean. It is true that one of this conditions is going to evaluate to true since price is either `<10`, or `>=10`, compiler does not know that. For that we need to add else statement and return a boolean there too.

Q4)(D)

Option B and option C are incorrect options since three overloaded methods compiles fine without issue. They are properly defined overloaded methods. The code does not compile because of the print statement in the `main` function, no argument version of the `nested()` method does not return anything, print method is not applicable for the void arguments.

Q5)(B)

Many programming languages allow passing parameters *by reference or by value*. In Java, we can only pass parameters *by value*. All object references in Java are passed by value. This means that a copy of the value will be passed to a method. But the trick is that passing a copy of the value also changes the real value of the object.

Q6)(C)

Getter methods should return the value, means that they have a return type. Setter methods should take an argument with the type of the variable that they are setting and does not return any value. Setter methods should start with the name ‘set’ and the getter methods should start with the name ‘get’.

Q7)(B)

If `this()` is used, it must be the first line of the constructor. If the no-argument `this()` is called, then the class must explicitly implement the no-argument constructor. If arguments are provided to `this()`, then there must be a constructor in the class able to take those arguments. `super()` and `this()` can not be used at the same time because it would lead to a double call to `super()` eventually leading double initialization.

Q8)(?)

There is no conversion between the object `Long` and `int`. Primitive `int` is autoboxed to the wrapper class `Integer` but an `Integer` object can’t be assigned to a `Long`. No correct answer.

Q9)(C)

Static variables belong to a class. They’re common to all instances of a class and aren't unique to any instance of a class. static attributes exist independently of any instances of a class and may be accessed even when no instances of the class have been created. You can compare a static variable with a shared variable. A static variable is shared by all the objects of a class.

Q10)(A)

All of the options compiles and runs with different results, but only the `this(4)` placed at the given position gives us the result we want. When the overloaded constructor is called with the primitive integer 4, boolean `outside` variable is **`false`**. We know that default value for non-initialized boolean is false. So, `outside ? rope : rope+1` ternary operator evaluates and returns the right-hand (rope + 1) side which is 5.

Q11)(B)

All of the options except option B is true. AN instance of the one class can not access package-private attributes in a parent class if the parent class is not in the same package.

Q12)(D)

None of the options inserted ensure the class data is properly encapsulated because the instance variable `stuff` is declared **`public`** in the first place. Instance variables must be declared **`private`** and appropriate getter and setter methods should be defined for them to ensure encapsulation.

Q13)(C)

Java will not insert a default no-argument constructor, if we define any constructor, which may be a no-argument constructor or not. A class cannot contain duplicate constructor. If a class extends another class that has only one constructor that takes a value, then the child class must explicitly declare at least one constructor because compiler cannot insert default no-argument constructor. We must write a constructor and call the parent class constructor ourself.

Q14)(A)

There can only be one variable argument in the method parameters and variable arguments should always be placed as the last parameter otherwise we will get a compilation error.

Q15)(C)

Only the `age` variable is changed because the copy reference in the method is used to access objects field. Assignments does not affect the original variables since the copies of them are passed to the method. Integer `speed` array is also left unchanged although it is accessed, because it is accessed after the reassignment to a new array.

Q16)(B)

Overloaded methods are methods with the same name but different method parameter lists. Overloaded methods can't be defined by only changing their return type or access modifiers or both. Overloaded methods must have method parameters different from one another.

Q17)(D)

Encapsulation says nothing about performance or concurrency, they're irrelevant. Some of the reasons to encapsulate the state of a Java object are as follows:

- To prevent an external object from performing dangerous operations
- To hide implementation details, so that the implementation can change a second time without impacting other objects
- To minimize the chance of coupling
- To increase usability (By keeping data private and providing public well-defined service methods the role of the object becomes clear to other objects. This increases usability.)

Q18)(A)

Arrays are objects so their references' copies are passed as arguments, using this reference, we can modify the object. Strings are objects too, but the problem is they're immutable and cannot be modified.

Q19)(B)

Although accessible using an instance, the static members are better accessed by prefixing their name with the class or interface names. .Option C would have been correct if if the method was imported using static import. By using static import, we can drop the prefix and just use the name of the static variable or method. Option A has invalid syntax. Option D is actually correct too but since the two classes are in the same package it is not required to specify package

Q20)(D)

Option A and B are incorrect since a method with a return type of `byte` or `String` or any other type must use `return` statement to return a value of that type. A method with a `void` return type can still use return statement to exit the method.

Q21)(C)

Final variables cannot be reassigned. So the method parameter `score` should not be **`final`** because it is essentially being reassigned with post-unary increment operator. Also `result` variable should not be **`final`** too because obviously it is being reassigned at the **`return`** statement.

Q22)(D)

super() is used to call a constructor in the parent class and super is used to reference a member of the parent class.

Q23)(B)

The given method is defined with the default access modifier which is also called package-private access modifier. Method or variables that have package-private access can only be accessed by classes in the same package that they are defined in.

Q24)(A)

Variable `strength` is defined with access modifier **`protected`** which means subclasses and classes from the same package in which the `strength` variable's enclosing class is defined can access the variable directly. So the variable's access modifier should be changed to **`private`**. Note that encapsulation is to disable unwanted direct access to data of a class from outside classes so it is not mandatory to define getter and setter method for `material` variable since it is already defined **privated**, hence encapsulated.

Q25)(A)

Method identifiers cannot start with numbers. Hyphen character cannot be used in a method identifier. **`new`** is a reserved keyword in java so it cannot be used as a method name either. Underscore and dollar symbol are allowed to be used in method identifiers and unlike numbers, they can be used as a first character in a method identifier.

Q26)(D)

There is no line of code that can be inserted to make the code compiled. Because the return type is placed wrong. Return type of a method must come right before the method name and after the modifiers (access or non-access) of the method. Having the method corrected option C would be the correct answer.

Q27)(B)

A change made to the data within an object passed to a method is reflected in the calling method since the reference of that object is copied when passing to the method but we are accessing the same object since it is pointing to the same object.

Q28)(D)

The code does not compile because the `contents` variable is declared final. If a variable is declared with the `final` keyword, its value cannot be changed once initialized. setContents() setter method tries to reassign the variable `contents` and that causes a compilation error. Having the error corrected the correct option would be option A.

Q29)(A)

In computing based on the Java Platform, JavaBeans are classes that encapsulate many objects into a single object (the bean). They are serializable, have a zero-argument constructor, and allow access to properties using getter and setter methods. The name "Bean" was given to encompass this standard, which aims to create reusable software components for Java. “is”, “get”, “set” are each JavaBeans method prefixes.

Q30)(C)

Option D is wrong because static import must be a field or member type. Option B would have been correct if it was a static import. Lastly, option A is wrong because the static and import keywords are placed reverse.

Q31)(D)

If we do not use any access modifier at all, that means the class member is package-private and called default access modifier, but note that there is no keyword default as a modifier, it is just the absence of access modifier.

Q32)(B)

The code does not compile, one line contains compilation error. We can only call `super()` from a constructor and even though it looks like a no-argument constructor because of the return type it is just another member method.

Q33)()

Static methods cannot reference an instance variable. We can’t reference instance variables from a static initialization block either. If the static variable declared as final, then we have to perform initialization explicitly whether we are using it or not and JVM won’t provide any

default value for the final static variable. For final static variable, it is compulsory that we should perform initialization before class loading completion.

Q34)(B)

The method return a short value or a value that can be automatically converted to short. Option D have a compile error since the assumed value is `int` so it needs to be casted to `short.` Even if it is corrected, long value cannot be converted to `short` without casting. Option B is the correct answer since `byte` can be automatically expanded to `short` safely.

Q35)(C)

Overloaded methods must have the same name or otherwise they would be different methods, not overloaded. Overloaded methods must have a different list of parameters, number of parameter can vary or if they have the same number of parameter type of the parameter should vary. Methods return type have nothing to do with the method signature in java, that is to say the overloading methods.

Q36)(B)?

Two lines must be removed for the class to be compiled. `monday` variable defined as `static final` but it is not initialized. That causes a compilation error. `wednesday` variable defined with no type at all. Variables should be defined with their type. `tuesday` variable is defined correctly.

Q37)(D)

The code does not compile because of line q3. We are invoking a constructor with one parameter but there is no such constructor. `Puppy(int wag)` is just a member method because it has a return type. Constructors do not have return types defined because they are inherently returning the class object itself.

Q38)(A)

A Java access modifier specifies which classes can access a given class and its fields, constructors and methods. The Java access modifier `public` means that all code can access the class, field, constructor or method, regardless of where the accessing code is located. The accessing code can be in a different class and different package. If a method or variable is marked as `private` (has the private access modifier assigned to it), then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.

Q39)(A)

The output of the following application is three. `sendHome()` method does not affect the original object since the copy reference is assigned to a newly created object, and then the size of this object is changed.

Q40)(B)

While accessing class members with instance reference is discouraged and considered as poor coding practice, it is legal to do so. Option A and Option D are correct and same. `this.Drink.water()` is meaningless since `Drink` is neither a class member nor an instance member that we could access with `this` reference.

Q41)(C)

Option B invokes the method with just an integer. Option A have the first two arguments right but after that it should take all strings as variable arguments but it is give an integer as third argument. Option D does not have an integer as a first argument. Option C is the only method call with the right arguments given to it.

Q42)(D)

Option A is true for static final variables, not for static variables. Option B is irrelevant since static and final modifiers does not set access, access modifiers does that. Option C is wrong since the opposite is true, it is possible to reference static methods using static imports. Option D is the only true statement which states that a static variable is always available in all instances of the class.

Q43)(A)

The first line of every constructor does not have to be a call to the parent constructor since it can be a call to `this()` with or without arguments.

Q44)(A)

None, because the code tries to reference instance final variable `height` in a static initialization block. Having corrected that issue, we wouldn't need to remove any final modifiers because all final variables are assigned in the initialization blocks.

Q45)(D)

The code does not compile. Java implements a call to `super()` as the first thing in the constructor, but there is no no-argument constructor defined in the parent class that we are extending. So we need to explicitly call a constructor from parent class.

Q46)(A)

There is trick here that needs careful attention. Normally, one would think that short type of the overloaded choose method would run, because **`byte`** would be expanded to **`short`**. But the expression given as argument to method returns a value of type **`int`** because, **`+`** operator automatically turns all lower types to **`int`** before addition.

Q47)(C)

The code does not compile because of line m1. `getScore` method takes in a `Long` as an argument but we are giving it an **`int`**. We should give it **`long`** because then **`long`** can be autoboxed to wrapper class `Long`. We can also give it directly a `Long` object.

Q48)(A)

Method names can include dollar sign, at the end, beginning or in between. Methods can include underscore also, but they can't include hashtag or backslash and cannot start with numbers while they can include them.

Q49)(B)

The protected access modifier provides the same access as the default access modifier, with the addition that subclasses can access protected methods and member variables (fields) of the superclass. This is true even if the subclass is not located in the same package as the superclass. According to that information option B is the correct answer.

Q50)(D)

None of the imports given in the options allows `Teller` class to compile because `withdrawal` and `deposit` methods are instance methods and cannot be used without an instance. `Teller` class uses them without an instance like they are static methods but this is not the case. We need an instance of the `Bank` class to use these methods or the these methods must be defined **`static`**. If these methods were static then option A would be the correct answer.