Homework 2

Q1) (A)

Code does not compiles because of invalid syntax. '**double num1, int num2 = 0**' syntax is not a valid syntax. '**int num2 = 0**' should be on another line itself (more precisely it should be another statement, we can use semicolon instead of comma but it counts as separate statements.) or the '**int'** keyword must be deleted, meaning **num2** is going to be an **double** too.

Q2) (D)

Code does not compile because we are trying to reference an uninitialized variable. Unlike class and instance variables, local variables are not initialized to a default value. The compiler checks to make sure that you have assigned a value before you use a local variable.If these variables were not local, it would have print '**nullmetal**' since default initialization value for String reference type is **null**.

Q3) (B)

The variables in Java can be categorized into two types: primitive variables and reference variables. Reference variables are also known as *object reference variables* or *object references*.

Local variables don't have default values, so local variables should be declared and an initial value should be assigned before the first use.

Instance variables have default values. For numbers, the default value is **0**, for booleans it is **false**, and for object references it is **null**. Since the String type is an reference type, it defaults to **null.**

Q4) (B)

Identifiers are names of packages, classes, interfaces, methods, and variables. Properties of valid identifiers are as follows:

- Can have unlimited length
- Can start with a letter (lower or uppercase), a currency sign, or an underscore
- Can use a digit (not at the starting position, **2blue** in the question is not a valid variable name.)
- Can use an underscore at any position
- Can use a currency sign at any position

For addition, There can not be a variable with the same name as Java keywords or reserved words. As these names suggests, they're reserved for specific purposes.

Q5) (B)

Java uses CamelCase as a practice for writing names of methods, variables, classes, packages and constants. Class names should be nouns, in mixed case with the first letter of each internal word capitalised. Interfaces name should also be capitalised just like class names.(FooBar)

- Methods should be verbs, in mixed case with the first letter lowercase and with the first letter of each internal word capitalised.(getData, applyChanges etc.)

- Variable names should be short yet meaningful. Should not start with underscore or dollar sign characters. Should be mnemonic, designed to indicate to the observer the intent of its use. One-character variable names should be avoided except for temporary variables.(fooBar)

- Constant variables should be all uppercase with words separated by underscores. There are various constants used in predefined classes like Float, Long, String. (FOO_BAR)

Q6) (C)

Primitive data types, as the names suggests, are the simplest data types in a programing language. Primitive variables and object reference variables differ from each other in multiple ways. The basic difference is that primitive variables store the actual values, whereas reference variables store the address of the objects they refer to. Since primitive types holds actual values and do not refer to objects we cannot call methods on them.

- On most platforms, integers and characters are examples of types that are primitive but can be boxed. Boxing means wrapping them in an object so they have the behavior of an object. Primitive types in java have corresponding wrapper classes(for example **Integer** for **int**, **Double** for **double** etc.).

Q7) (C)

In Java SE 7 and later, any number of underscore characters (_) can appear anywhere between digits in a numerical literal. This feature enables you, for example, to separate groups of digits in numeric literals, which can improve the readability of your code. You can place underscores only between digits; you cannot place underscores in the following places:

- At the beginning or end of a number
- Adjacent to a decimal point in a floating point literal
- Prior to an F or L suffix
- In positions where a string of digits is expected

Q8) (C)

Wrapper classes are those objects wraps a primitive data type within them. In the java.lang package java provides a seperate class for each of the primitive data types namely Byte, Character, Double, Integer, Float, Long, Short. Wrapper classes provide a way to use primitive data types as objects. Sometimes we must use wrapper classes, for example when working with Collection objects, such as **ArrayList**, where primitive types cannot be used.

Q9) (C)

The code does not compile because there is no class such as **integer** in java, it should be **Integer** instead. Primitive type is **int** and the corresponding wrapper class type is **Integer**.

Q10) (C)

When declaring a class in java, we are just creating a new data type. A class provides the blueprint for objects. Objects can be created from a class. However obtaining objects of a class is a two-step process:

- Declaration : Declare a variable of the class type. Variable does not define an object. Instead, it is simply a variable that can *refer* to an object.

- Instantiation and Initialization: Must acquire an actual, physical copy of the object and assign it to that variable. We can do this using the ***new*** operator. The ***new*** operator **instantiates** a class by dynamically allocating(i.e, allocation at run time) memory for a new object and returning a reference to that memory. The ***new*** operator is also followed by a call to a class constructor, which initializes the new object. A constructor defines what occurs when an object of a class is created.

Q11) (D)

In Java we have two primitive floating point types: float and double. The float is a single precision type which store numbers in 32 bits. The double is a double precision type which store numbers in 64 bits. Floating point numbers with an ‘F’ or ‘f’ suffix are of type float, double numbers have ‘D’ or ‘d’ suffix. The suffix for double numbers is optional. Assignment contexts allow the use **widening primitive conversion**. Widening is done without doing any explicit casting and hence we can call it as “Auto widening” or “Implicit widening”. In the case of narrowing (assigning larger type to smaller type) explicit typecasting is required.

19 specific conversions on primitive types are called the widening primitive conversions:

…
- int to long, float, or double
- long to float or double

- float to double

Q12) (A)

Byte is the smallest value which is an 8 bit integer value, chars are 16 bit Unicode values, floats are 32 bit decimal values, double is the type for double precision 64 bit decimal values.

Q13) (D)

There is no specific order to constructors, instance variables and methods names to appear. They can be declared in any order.

Q14) (B)

The code doesn't compile because it tries to declare multiple variables of different types in the same statement. The shortcut to declare multiple variables in the same statement only works when they share a type.

We can declare and initialize multiple variables in the same statement. We can declare many variables in the same declaration as long as they are all the same type. We can also initialize any or all of those values inline.

Q15) (C)

The code block with the static modifier signifies a *class* initializer; without the static modifier the code block is an *instance* initializer. Static initializer is automatically invoked when the class is loaded, and there's no other way to invoke it. Instance initializers are executed in the order defined when the class is instantiated, immediately before the constructor code is executed, immediately after the invocation of the super constructor.

Q16) (A)

There is no **default value** for **local variables**, so **local variables** should be declared and an **initial value** should be assigned before the first use. If they were instance variables for example, double, short and int would the default value of **0**.

Q17) (A)

Finalize() is the method of Object class. Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.

The general contract of finalize is that it is invoked if and when the Java virtual machine has determined that there is no longer any means by which this object can be accessed by any thread that has not yet died, except as a result of an action taken by the finalization of some other object or class which is ready to be finalized. The finalize method may take any action, including making this object available again to other threads; the usual purpose of finalize, however, is to perform cleanup actions before the object is irrevocably discarded. For example, the finalize method for an object that represents an input/output connection might perform explicit I/O transactions to break the connection before the object is permanently discarded.

The finalize method is never invoked more than once by a Java virtual machine for any given object.

Q18) (D)

String is not a wrapper class, simply because there is no parallel primitive type that it wraps. The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
A *(primitive) wrapper class* in Java is one of those eight classes that wrap a (=one) primitive value. String *wraps* a char[] so according to this it is *not* a (primitive) wrapper class.

Q19) (C)

Code creates a linked list, **link1** being the first link and **link3** as the last in the link. Then changes the references so that link2 and link3 points to each other, creating circular linked list. Then nulling the link1 and link3 references. Since there is no object pointing to '**x**' object anymore it is no more accessible and eligible for garbage collection.

Q20) (C)

Decimal types in java are **float** and **double** but if no **F** suffix is added as last letter to number, java interpret is as double, which is to say default decimal in java is double type.

Q21) (B)

Class names can be used as variable names, although it is discouraged. But **int** is a reserved keyword in java and can not be used as a variable name.

Q22) (B)

bar is certainly an instance variable, if it was a method call it would have parenthesis right to the name. **foo.bar** can be used to read or write to bar if it has been declared with sufficient access modifier. **bar** is certainly not a local variable because they're just scoped to the method they're defined in and hence cannot be accessed from outside.

Q23) (C)

Identifier rules applies to class names as they are identifiers too. Underscore, dollar sign and numbers are all valid in identifiers and can be used but numbers can't be used as a first letter of an identifier.

Q24) (D)

Underscore is allowed in a numeric literal but can't appear just before or next to point character. Only can appear between to numbers. Having the underscore corrected, correct answer would be option B because we see that new keyword on the right hand side which is used to instantiate objects so we should use wrapper class **Double** not primitive. Java then would unbox that object because we set the variable type as primitive (**double**).

Converting an object of a wrapper type to its corresponding primitive value is called unboxing. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

Q25) (D)

Local variables do not have default values. Code would compile if it is not referenced before being set to a value. If the variable was an instance variable, it would have default to **null** since **String** is an object and objects without initialization defaults to null in java.

Q26) (C)

Instance variables initialized with default values if they are not initialized with a value. For decimal types **double** and **float** it is 0.0 and for integer types (**byte**, **int**, **short**, **long**) it is **0**.

Q27) (B)

Since primitives are not objects we can't call methods on primitives. With autoboxing we can convert a primitive to a wrapper class object simply by assigning it. valueOf() method is public static method for wrapping primitives into objects, to convert a wrapper class object to primitive we need to use corresponding (**xxxValue()** (**intValue()**, **doubleValue()** etc.) method on wrapper object. We can't store a primitive directly into ArrayList since it does not

allow primitive types to be used. At first it looks like we can store them but behind the scenes java implicitly converts primitives to corresponding wrapper class (autoboxing).

Q28) (C)

Java allows assignment of Integer objects to int primitives and vice versa via autoboxing mechanism. We cannot call instance methods on primitives since they are not objects to begin with.

Q29) (D)

We need to call the constructor since in the constructor code prints ‘bounce’. The **new** keyword is a Java operator that creates the object. The **new** operator is followed by a call to a constructor, which initializes the new object. The **new** operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the object constructor. The **new** operator requires a single, postfix argument: a call to a constructor. The name of the constructor provides the name of the class to instantiate.

Q30) (B)

III, is a valid syntax but it does not assign cat variable. IV, is not a valid syntax, declarations should different statements or only one type keyword must be present at the beginning. The shortcut to declare multiple variables in the same statement only works when they share a type. I and II both assigns. II uses two statements because of statement terminator (**;**) is used.