Bünyamin AKTAŞ
25/04/2020

ASSIGNMENT – 2

Q1 **Answer is A.**

In Java, multiple variables can be declared in a single statement if they share the same data type. In **B**, two variables of type *int* are declared, none of them are initialized but the statement compiles. In **C**, two variables of type *int* are declared and one of them is initialized. This statement compiles, too. In **D**, two variables are declared and both are initialized; this also compiles. In **option A**, two different data types are used so this statement does not compile.

Q2 **Answer is D.**

In Java, instance and class variables do not require to be initialized since they have a default value. But local variables need an explicit initializing. In this code snippet, *chair* variable, which is a local variable, is not initialized so the code does not compile. The correct option is **D.**

Q3 **Answer is B.**

All instance variables of class types in Java defaults to *null*. So type *String* also initialized as *null* by default. Answer is **B.**

Q4 **Answer is B.**

Identifier names can begin with a letter, $ and _. A number cannot be used at the beginning but it can be put after beginning. So the **the option B** beginning with the number *2* is not valid.

Q5 **Answer is B.**

For methods and variables, most Java developers start names with a lowercase letter followed by CamelCase. For class names, they use uppercase followed by CamelCase. In this question **option B, FooBar** best follows standard Java naming conventions.

Q6 **Answer is C.**

toString() is an instance method that can be invoked on objects. It cannot be invoked on primitive data type variables. *Integer* and *Object* are classes so that their instances can use *toString()* method while *int* type variables cannot. So the answer is **Two, C.**

Q7 **Answer is C.**

Numerals can have underscores in order to increase readability. But they cannot start with an underscore. So *999* and *9_9_9* are valid definitions while *_9_99* is not. So **the option C, int num = _9_99** does not compile.

Q8 **Answer is C.**

Wrapper classes are object data types of their corresponding primitive types. For example primitive data type *int* has a corresponding wrapper class called *Integer.* These are used when the value we need, must be in object data type.

A – *int* is a primitive data type, not a class.

B – There is no such thing as *Int*, neither a class nor a data type.

C – Integer is a wrapper class corresponding to primitive data type *int.*

D – Object is a class but not a wrapper one.

Q9 **Answer is C.**

There is neither a primitive data type nor a class called *integer* which begins with a lowercase i. If capital I were used or primitive data type *int* were used then it would be valid. And *a* would be 2, since it is reassigned in *main()* method, *b* would be 3 and the code would print out 5.

But this way, the code does not compile. The answer is **C**.

Q10 **Answer is C.**

When *new* keyword is used, a new object is instantiated by allocating a memory and a reference to that memory is bound to the new object. So, the option best explaining what *new* does, is **C**.

Q11 **Answer is D.**

Primitive data type *float* needs an f following the number to be valid, while *double* doesn't need that. So *p2* and *p3* are valid. Java also allows widening which means a data type can be transformed into a wider data type such as *int* → *long*, *float* → *double*. So p1, which transforms a *float* data to *double*. *p1* is also correct. But *p4*, which is missing *f* at the end, would trigger a compiler error. So the correct answer is **D.**

Q12 **Answer is A.**

A *byte* is 8 bits, a *char* is 16 bits, a *float* is 32 bits and a *double* is 64 bits. So the answer is **A.**

Q13 **Answer is D.**

Instance variables (fields) and methods can appear anywhere inside a class. A constructor is actually a method. So there is no specific order rule. The answer is **D.**

Q14 **Answer is B.**

x1 is a valid statement so there is no compiler error.

x2 has multiple data types in one declaration which is not allowed. So here is a compiler error.

x3 is using a variable that is not initialized. This line is also not valid but the compiler would throw out an error on *line x2*, first. So the answer is **B.**

Q15 **Answer is C.**

When a code snippet is put between braces, outside of any method, this code snippet is called *instance initializer,* which would run before constructors. So *line 2* and *line 7* are instance initializers. *Line 3 – line 5* are constructor. Finally, *line 5* has a static keyword so, it is a *static initializer* which will run when the class is first used. So here we have **two instance initializers that makes option C true**.

Q16 **Answer is A.**

As explained in **Q2** local variables does not get a default value and they must be initialized explicitly. Since the variable *defaultValue* is in main() method so is a local variable, it must be initialized to a value. So the code does not compile and the answer is **none, A**.

Q17 **Answer is A.**

finalize() method can be implemented in objects. If Java Garbage Collector(GC) decides to collect the object, it calls this method. If GC never runs, the method never gets called. But if, for some reason, GC fails to collect the object, it will never call the *finalize()* method a second time. This method is used when it is necessary to ensure some tasks be performed before releasing the memory used by the object. So, the answer is that **it may be called zero or one times, A.**

Q18 **Answer is D.**

Wrapper classes are object data types of their corresponding primitive types. For example primitive data type *int* has a corresponding wrapper class called *Integer.*

Double has a corresponding primitive data type called **double.** So this is a wrapper class.

Long has a corresponding primitive data type called **long.** So this is a wrapper class.

String does not have any corresponding primitive data type. So it is not a wrapper class. The answer is **D.**

Q19 **Answer is C.**

With lines 18-19, *link2* and *link3* are made to point each other. Then on lines 20-21 *link1* and *link3* are made null that wipes out their references. So with *link1* and *link3* none of the objects created is accessible. *link2* has reference to the object named “y” which has reference to the object named “z”. And the object “z” has reference to the object “y” again. So the answer is **C.**

Q20 **Answer is C.**

Primitive data types *byte* and *short* does not hold numbers with decimal points. They only hold integers. *float* can hold decimals but the statement needs an *f* at the end. So only ***double*** can hold the value in the statement. The answer is **C, double.**

Q21 **Answer is B.**

Java has reserved words that are not allowed to be used as variable names. All primitive data types are reserved words.

k1 *Integer* is a class name but it is not forbidden to use it as a variable name. So the statement is valid.

k2 *int* is a reserved word so this statement is not valid. So this is the first line not to compile.

k3 the variable *Integer* is incremented. So this statement is valid.

k4 is trying to increment a variable but it is a reserved word. So this is not valid, too. But the question is which line is the first not to compile. So the answer is **k2, B.**

Q22 **Answer is B.**

If *foo* is an instance than *bar* in *foo.bar* could only be an instance variable in *foo.* So it can be used to read the value or to write a value if it is not private. But it cannot be a local variable since local variables cannot be accessed outside of their code block. So the answer is **B.**

Q23 **Answer is C.**

As explained in **Q4** identifier names cannot begin with a number. So **5MainSt** is not valid for a class name, the answer is **C**.

Q24 **Answer is D.**

The value in the statement is a *double* type decimal value. But it has an underscore before dot (.). This is not allowed, the underscore can only be between digits. So this code never compiles. The answer is **D.**

But if the underscore is removed then (double, Double) pair would make the code compile. Because *new* keyword is used to create an object the second keyword must be the name of the class. But first keyword can be primitive data type *double.*

Q25 **Answer is C.**

Local variables don't get any default value. They need an explicit initialization. But the initialization doesn't have to be on the declaration line. It can be later as well. So the answer is **C**.

Q26 **Answer is C.**

For instance variables, *int*, *long* and *short* types defaults to 0. But *double* type defaults to 0.0. So the answer is **Three, C**.

Q27 **Answer is B.**

A – Methods cannot be called on primitives. They can only be called on objects.

B – A primitive can be converted to its corresponding wrapper class object simply by assigning it. This is the answer.

C – *valueOf()* convert String data to numerals. This is not about wrapper classes.

D – Primitives cannot be stored into ArrayList and this is one of the reasons of that wrapper classes exist.

Q28 **Answer is C.**

On the last line a method is called on a primitive and it is not allowed in Java, although a wrapper object can be converted into a primitive so the line before the last is valid. So **the code does not compile**. The answer is **C**.

Q29 **Answer is D.**

Constructor are technically methods but *new* keyword must be used when they are called. So the correct code must begin with *new* and continue as calling a method like *new constructor()*. So the answer is ***new TennisBall()*** **D**.

Q30 **Answer is A.**

I – *animal* is assigned to both variables correctly on this statement.

II – This line has two different statements but the second one is missing *String* class name. This code does not compile.

III – On this statement only the variable *dog* is assigned a value.

IV – On this statement there are two declarations which is not allowed in Java. So this code cannot compile.

The only statement that does what is expected is the **I, A**.

Q31 **Answer is C.**

Wrapper class for *byte* is *Byte*. Wrapper classes for *char* and *int* are *Character* and *Integer.* So the **option C** is correct that these two primitives' wrapper classes are not merely the name of them with an uppercase letter.

Q32 **Answer is A.**

A – *String* variables can be set to *null* whether they are instance or local variables.

B – If *String* variables declared as *public* or *protected* they can be set outside of the class they are defined in.

C – It is not obligatory that instance variables be set in the constructor. They can be set in a method.

D – If the instance variable is not final they can be set anytime.

Q33 **Answer is A.**

A – Primitive types begin with a lowercase letter in Java.

B – Primitive types cannot be set to null. Objects can be set to null.

C – *String* is not a primitive, it is a class.

D – Custom primitives cannot be created in Java.

Q34 **Answer is D.**

Although we can signal to JVM that it is proper time to run garbage collector with *System.gc()* method, it is not guaranteed that JVM will agree to it. So there is no way to force garbage collection to start. The answer is **D**.

Q35 **Answer is C.**

When *fruit3* is assigned to *fruit1* it points to *apple*. Then *fruit2* is assigned to *fruit3* which points to *apple* so *fruit2* points to *apple*, too. Then *fruit1* is assigned to *fruit2* and it starts to point to *apple* through *fruit2* no matter it already points to *apple.* So because the all references point to *apple,* two unnecessary objects can be collected by garbage collector. The answer is **C.**

Q36 **Answer is B.**

After *new* keyword shall come the constructor of *Double* wrapper class which begins with an uppercase letter. Data type in declaration can be primitive type *double* or the wrapper *Double* due to that primitives and their corresponding wrapper classes are interchangeable. **A** and **C** has *double* with a lowercase letter at the beginning for the second blank. So these options are wrong. The option **B** has what we need, **[double or Double], Double**.

Q37 **Answer is B.**

When an object is created **instance variables** are initialized firstly. So the value of variable *first* is "instance" firstly. Then **instance initializers** run. This time value of *first* becomes "block". Then **constructor** run and value of *first* becomes "constructor". So after creation of the object, which also means calling and ending the constructor, the value of *first*, to be printed, is "constructor". The answer is **B.**

Q38 **Answer is C.**

null cannot be assigned to primitives. It can assigned to objects. *int* is a primitive type while *Integer* is a wrapper class and *String* is a class. So *int i = null;* does not compile. The other **two compiles.** The answer is **C.**

Q39 **Answer is C.**

Static members cannot call an instance member without creating an object. But instance members can call instance and static members since static members do not require any instance of the class. So instance variables can be called only from instance methods while static variables can be called from both instance and static methods. The answer is **C.**

Q40 **Answer is B.**

Numerals in Java can have underscores between digits in order to increase readability. So decimal numbers in option **A** and **C** which has underscore between digits are valid. But the numeral in option **B** has an underscore between a dot and a digit. It is not a valid initialization. The statement in **B** does not compile.

Q41 **Answer is A.**

Primitive data type *byte* is 8-bits, *short* is 16-bits, *int* is 32-bits and *long* is 64-bits. So the order from smallest to largest data type is ***byte, short, int, long***. The answer is **A**.

Q42 **Answer is A.**

Java uses dot (.) to access attributes of an object. Since the field *name* is public it is valid to use *cat.name* to access to it. "-" is not used in Java. *Cat* class does not have any attribute called *setName. []* annotation is used to access to array elements not to access to object attributes. So the answer is **A.**

Q43 **Answer is B.**

If the method *finalizer()*'s name were *finalize()* it may be called by garbage collector. But it is not, it is a normal method that does not get called automatically. So only the *play()* methods print something on screen. When *car.play()* method is called, it will print *play-*. When *doll.play()* method is called, it will print *play-*, too. So the output will be ***play-play-*** The answer is **B.**

Q44 **Answer is A.**

Notations in **B** and **C** are not valid in Java so they do not compile. Statement in **A** is valid it assign an *int* value to the attribute of object *p,* not to field in the current class. Primitive data type *int* can be assigned to a *double* variable since *double* is wider than *int.* This method is usually used to call without creating an object, since it is static, and to set a new value to any object's beakLength attribute. So the answer is **A.**

Q45 **Answer is B.**

Autoboxing means that Java can convert a primitive data to its corresponding wrapper class automatically when it is needed. *parseInt()* method returns a primitive data while *valueOf()* returns a wrapper class object. So in order to avoid autoboxing, variable *first* must be declared as primitive *int* because *parseInt("5")* will return a primitive *5* and variable *second* must be declared as wrapper class *Integer* because *valueOf("5")* will return a wrapper class object. So the answer is **B, int, Integer**.

Q46 **Answer is B.**

There are 3 objects created; *elena, diana* and *zoe. elena* object has reference to *diana* object at first but later its reference changes to *zoe* object. So there are 2 object in use, *elena* and *zoe.* The remaining one, *zoe* can be collected by garbage collector. The answer is **One, B.**

Q47 **Answer is C.**

Constructors are methods without a return type and that have same name of their class.

A – This is a static method and *TennisBall* should not be in definition. It is class name not a keyword to be used in definitions.

B – This is also a static method that should not have *TennisBall* in its definition and needs a return type. Also *return new TennisBall():* is not valid because there is no use for colons (:) in Java.

C – This statement is valid for a constructor definition since it has same name with the class and has not a return type.

D – This is a normal method that has a return type. Having same name with the class does not make this method a constructor.

Q48 **Answer is A.**

Contrary to **Q43** this code has a method name *finalize()*. This method would be called automatically by Garbage Collector(GC) if it runs. *System.gc()* method signals GC to run but it may not run. So for that *play()* method of two objects is called so two "play-" will be printed out. But *finalize()* method may be called once for each object or may never be called. So "clean-" may be printed out zero, one or two times. **Option A saying only one "play-" can be printed is not correct. So the answer is A.**

Q49 **Answer is B.**

Every wrapper class has a constructor that takes primitives as parameter and converts them to wrapper object. The methods mentioned in options **A, C** and **D** do not exist. So the answer is **B.**

Q50 **Answer is C.**

In main method constructor of Sand class is called. The constructor prints out "a". Then *run()* method of Sand class is called and that method calls Sand's constructor again printing out "a" again. The method calls *Sand()* method finally which prints out "b". So the output will be "aab". So the **option C** is the answer.