Emre KARAGÖZ

ASSIGNMENT 2
JAVA DATA TYPES
(HW 2)

Q1-) Option A

"`double num1`" and "`int num2`" are different types of variables. We cannot declare different types of variables as same part of a declaration in Java so, **Option A** is not a valid declaration and it does not compile.

Q2-) Option D

```
String chair, table = "metal";
```

The code does not compile because the local variable "`chair`" is not initialized here. The compiler cannot read an uninitialized value and generates this error: "variable chair might not have been initialized". **Option D** is the answer.

Q3-) Option B

`String` is not a primitive data type in Java, it is a reference data type and all reference data types defaults to null. So, **Option B** is the correct answer.

Q4-) Option B

- In Java, all variable names must begin with a letter of the alphabet, an underscore (_), or a dollar sign ($). The convention is to always use a letter of the alphabet. The dollar sign and the underscore are discouraged.
- After the first initial letter, variable names may also contain letters and the digits 0 to 9. No spaces or special characters are allowed.
- The name can be of any length, but don't get carried away. Remember that you will have to type this name.
- Uppercase characters are distinct from lowercase characters. Using ALL uppercase letters are primarily used to identify constant variables. Remember that variable names are case-sensitive.
- You cannot use a java keyword for a variable name.

According to information above, "`_blue`", "`blue$`" and "`Blue`" are valid names but "`2blue`" is not a valid variable name because it starts with a number. **Option B** is the correct choise. **https://mathbits.com/MathBits/Java/DataBasics/Namingrules.htm

Q5-) Option B

In Java, class names should be nouns, as they represent “things” or “objects.” They should be mixed case (camel case) with only the first letter of each word capitalized, as in the following:

```
public class FooBar{...}
```

Option B follows standard Java naming conventions, it is the answer.

**https://www.oreilly.com/library/view/java-8-pocket/9781491901083/ch01.html

Q6-) Option C

“String” and “Integer” are objects but “int” is a primitive type. Objects have methods, yet primitives do not have. The first method does not compile because “value” is an int so, we cannot call the convert method there. In the second and third methods, “value” is an object, it’s a reference type. We can call a method on “value” in these methods since it is of a reference type and, these methods compile. The answer is **Option C**.

Q7-) Option C

We can add underscores anywhere except at the beginning of a literal, the end of a literal, right before a decimal point, or right after a decimal point.

```
int num = _9_99
```

There is an underscore at the begining of the literal, **Option C** does not compile.

Q8-) Option C

Each primitive type has a wrapper class, which is an object type that corresponds to the primitive. The table below lists all the wrapper classes along with the constructor for each.

Primitive type	Wrapper class	Example of constructing
boolean	Boolean	`new Boolean(true)`
byte	Byte	`new Byte((byte) 1)`
short	Short	`new Short((short) 1)`
int	Integer	`new Integer(1)`
long	Long	`new Long(1)`
float	Float	`new Float(1.0)`
double	Double	`new Double(1.0)`
char	Character	`new Character('c')`

So, Integer is a wrapper class, **Option C** is the answer.

Q9-) Option C

There is not a class called "integer" in java. There is “int” as a primitive type and there is “Integer” as a class. Because of this, the code does not compile. To compile it, we can change the class name “integer” to “Integer”. Answer is Option C.

Q10-) Option A

The answer is **Option A**. The Java “`new`” keyword is used to create an instance of the class. In other words, it instantiates a class by allocating memory for a new object and returning a reference to that memory. We can also use the new keyword to create the array object.

*https://www.javatpoint.com/new-keyword-in-java

Q11-) Option D

Option A:

`double d1 = 5f;` // p1 , this float value is assigned to double and p1 compiles.

Option B:

`double d2 = 5.0;` // p2 is valid and it compiles

Option C:

`float f1 = 5f;` // p3 is valid and it compiles

Option D:

`float f2= 5.0;` // p4 does not compile, it need the suffix f. This is the answer.

Q12-) Option A

Keyword	Type	Example
boolean	true or false	true
byte	8-bit integral value	123
short	16-bit integral value	123
int	32-bit integral value	123
long	64-bit integral value	123
float	32-bit floating-point value	123.45f
double	64-bit floating-point value	123.456
char	16-bit Unicode value	'a'

The table above demonstrates Java primitive types.

The list in **Option A** is `byte(8-bit)`, `char(16-bit)`, `float(32-bit)`, `double(64-bit)` and this is the correct order from smallest to largest.

Q13-) Option D

Element	Example	Required?	Where does it go?
Package declaration	`package abc;`	No	First line in the file
Import statements	`import java.util.*;`	No	Immediately after the package
Class declaration	`public class C`	Yes	Immediately after the import
Field declarations	`int value;`	No	Anywhere inside a class
Method declarations	`void method()`	No	Anywhere inside a class

The table above demonstrates elements of a class.

Option A: Constructor, instance variables, method names
Option B: Instance variables, constructor, method names
Option C: Method names, instance variables, constructor

There no rule to order the instance variables, constructor, and method names. They can be in any order within a class declaration. All these orders are valid and the answer is **Option D**.

Q14-) Option B

"line x2" does not compile and throws an error because, Java does not allow multiple data types to be declared in the same declaration. To compile this code, we can declare variables "hot" and "cold" separately from each other and assign a value to variable hot. The answer is **Option B.**

Q15-) Option C

Sometimes code blocks are inside a method. These are run when the method is called. Other times, code blocks appear outside a method. These are called *instance initializers*.

```
public class Q15 {
    { System.out.println(); }        //instance initializer
    public void Bowling () {         // constructor
        System.out.println();
    }
    static { System.out.println(); } // static initializer
    { System.out.println(); }        // instance initializer
}
```

There is two instance initializer here, the answer is Option C.

Q16-) Option A

`defaultValue` is a local variable. Local variables must be initialized but it is not initialized here. Because of this, the code is not compiled regardless of the variable type. **Option A** is the answer.

Q17-) Option A

Java allows objects to implement a method named `finalize()` that might get called. This method gets called if the garbage collector tries to collect the object. If the garbage collector doesn't run, the method doesn't get called. If the garbage collector fails to collect the object and tries to run it again later, the method doesn't get called a second time.

According to this, **Option A** is the answer. `finalize()` method may be called zero or one times.

Q18-) Option D

Each primitive type has a wrapper class, which is an object type that corresponds to the primitive. There no primitive type named "`string`" in Java. "`String`" is a class but not a wrapper class. **Option D** is the answer.

Q19-) -

Q20-) Option A

"`byte`" and "`int`" are used for numbers without decimal points. "`float`" and "`double`" are used for floating-point (decimal) values but "`float`" requires the letter "f" following the number, so Java knows it is a float.

```
_______ pi = 3.14;
```

"`byte`" can fill this blank, **Option A** is the answer.

Q21-) Option B

```
int Integer = 0; // k1 valid but not preferable because, "Integer" is name of a wrapper class
integer int = 0; // k2 there is no type named "integer" and int is a name of a primitive type, not valid
Integer ++;      // k3 "Integer" is name of a wrapper class, this is not valid
int ++;          // k4 "int" is a name of a primitive type, not valid
```

k2 is the first line to not compile, **Option B** is the answer.

Q22-) Option B

The Java Dot Notation provides full access to instance variable. We can use it for both reading from and writing to them, but it cannot be used for local variables. According to this information, bar is not a local variable. **Option B** is the answer.

** http://jscheme.sourceforge.net/jscheme/doc/javaprimitives.html

Q23-) Option C

The language specification states that a class name should be a sequence of so-called Java letters or Java digits. The first character should be a Java letter. A Java letter is a set of Unicode characters, underscore (_), and dollar sign ($). A Java digit is a collection of numbers from 0 to 9.

`building, Cost$, _Outside` are valid class names, yet `5MainSt` is not valid. It starts with a digit, it must start with a name. **Option C** is the answer.

** http://dolszewski.com/java/java-class-naming-ultimate-guideline/

Q24-) Option D

```
_______d = new_______ (1_000_000_.00);
```

As explained in Q7, we can add underscores anywhere except at the beginning of a literal, the end of a literal, right before a decimal point, or right after a decimal point. there is an underscore right before the decimal point here, so this code does not compile. **Option D** is the answer.

Q25-) Option C

As mentioned in Q3 `String` is not a primitive data type in Java, it is a reference data type and all reference data types defaults to null. But the question asks about the local variable `String` and, local variables do not have a default initialization value. So, **Option C** is the answer.

Q26-) Option C

`defaultValue` is an instance variable. Instance variables are automaticaly initialized to their default values. The default value is “0” for “`long`”, “`short`” and “`int`” yet, it is 0.0 for double. But the question asks about the output “0” and answer **Option C** (three).

Q27-) Option B

In Java we cannot call methods on a primitive, we can only call methods on objects. This makes **Option A** is incorrect. "`ValueOf()`" method is used for converting to a wrapper class from a String, not a wrapper class object to a primitive. So **Option C** is incorrect. You cannot store a primitive directly into an `ArrayList` because, primitives were converted to objects before adding to an `ArrayList`. Namely, **Option D** is incorrect. **Option B** tells us we can convert a primitive to a wrapper class object simply by assigning it. This is an example of autoboxing and it is true.

Q28-) Option C

```
System.out.print(i.byteValue());
```

The code does not compile because of this line. "`i`" is a primitive cannot call methods on a primitive.

Q29-) Option D

To to call a constructor, we must use the "`new`" keyword and methods connot be called without paranthesis. **Option D (**`new TennisBall();`**)** compensates these requirements, it is the correct answer.

Q30-) Option A

I. **`String cat = "animal", dog = "animal";`**

- valid declaration

II. **`String cat = "animal"; dog = "animal";`**

- invalid, type is missing for dog

III. **`String cat, dog = "animal";`**

- valid but no value is assigned to cat

IV. **`String cat, String dog = "animal";`**

- invalid because type should be used once in a declaration

I. correctly assigns animal to both variables, **Option A** is the answer.

Q31-) Option C

As we can see from the table in Q8, wrapper class for "`int`" is "`Integer`" and the wrapper class for "`char`" is "`Character`". These are not merely the name of the primitive with an uppercase letter, the are new words. So, **Option C** (`char` and `int`) is the answer.

Q32-) Option A

As mentioned in Q3 and Q25 `String` is not a primitive data type in Java, it is a reference data type and all reference data types defaults to null. **Option A** is the true about `String`.

Q33-) Option A

Option A: Primitive types begin with a lowercase letter.

- This statement is true. All primitive types start with a lowercase letter.

Option B: Primitive types can be set to null.

- This statement is false. Reference types can be set to null, but primitive types cannot be set.

Option C: String is a primitive.

- This statement is false. `String` is not a primitive data type in Java, it is a class.

Option D: You can create your own primitive types.

- This statement is false. You can create your own classes, but not primitives.

Q34-) Option D

Java provides a method called "`System.gc()`". This method **suggests** that now might be a good time for Java to kick off a garbage collection run. Java is free to ignore the request. We cannot force garbage collection to occur at a certain point. Answer is **Option D**.

Q35-) Option C

Two of the `String` objects are eligible for garbage collection right before the end of the `main` method because, all of these references point to `String apple.` The answer is **Option C.**

Q36-) Option B

We can only call a constractor with a class name. So the first blank should be "`Double`". After the creation a "`Double object, it` can be assigned to either a "`double`" or "`Double`" owing to auto boxing. Namely, there 2 declarations that we can use here:

- `Double d = new Double(1_000_000.00);`
- `double d = new Double(1_000_000.00);`

The second one makes **Option B** the answer.

Q37-) Option B

Fields and instance initializer blocks are run in the order in which they appear in the file. The constructor runs after all fields and instance initializer blocks have run. So, "`constructor`" is assigned to "`first`" and it is the output. **Option B** is the answer.

Q38-) Option C

```
1. int i = null;

2. Integer in = null;

3. String s = null;
```

"`String s`" and "`Integer in`" are objects (reference type) and they are allowed to have a null reference which means they do not currently refer to an object. Primitive types will give you a compiler error if you attempt to assign them "`null`". In this question, "`int i`" cannot point to null because it is of type "`int`", it is a primitive. Line 1 does not compile but line 2 and line 3 compile. **Option C** is the answer.

Q39-) Option C

Type	Calling	Legal?	How?
Static method	Another static method or variable	Yes	Using the classname
Static method	An instance method or variable	No	
Instance method	A static method or variable	Yes	Using the classname or a reference variable
Instance method	Another instance method or variable	Yes	Using a reference variable

A static method cannot call an instance variable. Static variables can be called from both instance and static methods. **Option C** is the answer.

Q40-) Option B

```
double num = 2._718;
```

As explained in Q7, we can add underscores anywhere except at the beginning of a literal, the end of a literal, right before a decimal point, or right after a decimal point. there is an underscore right after the decimal point here, so this line does not compile. **Option B** is the answer.

Q41-) Option A

The table in Q12 demonstrates Java primitive types. The list in **Option A** (`byte(8-bit)`, `short(16-bit)`, `int(32-bit)`, `long(64-bit)` is in the correct order from smallest to largest.

Q42-) Option A

"`String name`" is an instance variable. We can use Java dot notation to reference instance variables in a class.

"`cat.name`" is the correct usage of dot notation, **Option A** is the answer.

Q43-) Option B

The code compiles and prints "`play-play-`" as the output. "`finalizer`" method is never used.

Q44-) Option A

Using Java dot notation is the most common way to implement this code. **Option A** (`p.beakLength = b;`) is a correct usage of dot notation. The other options do not compile, because braces ([ ]) are for arrays, not for instance variables such as `double beakLength`.

Q45-) Option B

The parseInt() methods returns a primitive, valueOf() method returns a wrapper class object. Autoboxing allows you to assign the return value to either a primitive variable or a wrapper class. To avoid using autoboxing, a primitive should be assigned to a primitive, a wrapper class object should be assigned to a wrapper class object directly. "int first" is a primitive variable and "Integer second" is a wrapper class object. **Option B** is the answer.

Q46-) -

Q47-) Option C

Option A:

- ```
  public TennisBall static create() {
        return new TennisBall();
        }
  ```

This is a static method, not a constructor.

Option B:

- ```
  public TennisBall static newInstance() {
        return new TennisBall():
        }
  ```

This is a static method, not a constructor. Besides, there must be semicolons instead of colons to compile.

Option C:

```
public TennisBall() {}
```

This is a valid constructor fort this class. It has the same name with class and don't have a return type.

Option D:

```
public void TennisBall() {}
```

This is not a valid constructor fort this class. Constructors don't have a return type.

Q48-) Option A

"`play-`" cannot be the output of this code, because `play()` is definitely called twice. **Option A** is the answer. Other options may be the output depending on whether garbage collector runs and depending on when it runs.

Q49-) Option B

Calling the constructor of the wrapper class converts a primitive to a wrapper class object without using autoboxing.

Q50-) Option C

First main "`main`" method calls the constructor and it outputs a. Then, "`run()`" method calls the constructor and it outputs "a" again. After that, run calls the "`Sand()`" method that outputs b. This code prints "aab", **Option C** is the answer.