ECE 325
Assignment 1 – Testing your understanding of the basics of Java

Jaskeerat Singh, 1687820

A1)
Value of the outcome: 200
Type of the outcome: int

Since both x and y are integers of value 100 each, integer addition in Java results in an integer outcome.

A2)
Value of the outcome: 200.0
Type of the outcome: double

When a double (y) is added to an integer (x), the result is converted to a double as to not lose precision in Java.

A3)
Value of the outcome: 1
Type of the outcome: int

When an integer is divided by another integer in Java, the fractional part of the answer (if there is any) is truncated and hence leaves only the integer part behind.

A4)
Value of the outcome: 1.0
Type of the outcome: double

When integer is divided by a double or vice versa, the result is cast to a double in Java in order to give a precise answer.

A5)
Value of the outcome: 219
Type of the outcome: int

When a character is added to another character the ASCII values of those two characters are added and cast into an int result. The ASCII value of ‘a’ is 97 and the ASCII value of ‘z’ is 122.

A6)
The value of c in the code snippet is -128 and c is of byte type.

A7)
This is not I would expect. I would have expected the value of c to be 128 instead of -128. This is happening due to the range of the byte data type, which ranges from -128 – 127. Since a + b is performed inside the bracket and then cast into type byte, overflow occurs since the value 128

cannot exist in byte type and hence circles back around to the minimum value the byte datatype can take, i.e, -128. If the value of byte b was 2 instead of 1, the resulting value of byte c would be -127 because it loops around to the minimum value of byte and then adds 1 to it. This happens because byte is limited by numbers stored in 8 bits in 2s complement binary.

A8)
Forcing the result of the addition to have a certain type in the third line of the code snippet is known as type casting. It is the assigning of one primitive data type to another. (W3 Schools)

A9)
It is necessary here to avoid a compile-time error, known as possible lossy conversion. This happens because java automatically converts byte + byte to int type in to prevent loss of data because of the limitations of byte data type. Us trying to force int back into byte type c results in this type of error and hence, it is needed to cast the int result of a+b into byte type ourselves to prevent the lossy conversion error. however, this does result in the unexpected answer of -128.

A10)
The five bugs I identified are listed below:

1. At the end of line 4 (return (i % 2) == 1), there is a semi colon missing, which would raise a compile-time error since it is a syntax mistake.
2. The comment on line 8 indicates that the loop should run 10 times but according to the for loop condition it will run 99 times resulting in a logical error. The for loop condition should instead be something as follows: for (int i = 1; i < 11 && isEven(i); i++).
3. In line 3, the type of the method isEven should be Boolean and not int since its purpose is to determine whether the integer fed into it is even, which is best represented by true or false, i.e, a Boolean type. Also, having it as int type will result in a compile time error because of type mismatch in line 9 (for(int i = 1; i < 100 && isEven(i); i++)) because it will be expecting a Boolean result for isEven but will be getting an integer.
4. There is a logical error in the method isEven() in line 4, this method actually checks if the integer fed into it is odd or not since odd numbers divided by 2 give a remainder of 1. It should be changed to: return (i%2) == 0;
   This will return true if the integer fed into it is even and return false if the integer fed into it is odd.
5. In line 9, there is a logical error since the for loop will exit prematurely as for it to continue, two conditions need to be met; i<11 AND isEven(i) should be true. However, since isEven(1) is false, the loop will exit in the first iteration itself, exiting prematurely. Instead of having it in the for loop condition, we should code it as follows:

```
for (int i = 1; i < 11; i++){
        if (isEven(i)){
                System.out.println(i);
        }
}
```

   This ensures that the code executes properly and goes through the full list of integers from 1 – 10.

After fixing these errors, the code should execute to give the correct output.