

## **Session 3**

**Student Name** \_\_\_\_\_

**Other Identification** \_\_\_\_\_

# **Data Storage and Its Implications**

In this laboratory you will:

1. Investigate the storage systems used for three primitive data types, `int`, `byte` and `double`.
2. Learn how these storage systems are reflected in the execution of Java programs.

## Data of Type Integer

Recall from Chapter 1 of *Computer Science: An Overview*, that data items of type integer are normally stored in memory by using two's complement notation and that this imposes a limit on the size of the values that can be represented. For example, with a two's complement system using only four bits for storage, the largest value that can be represented is seven. Of course, if this was the largest integer your machine could represent, it would be of little use. Therefore, computer systems use more than four bits for integer storage and can store rather large values. But, limitations still exist.

In the Java system, the maximum and minimum values of the numeric types are represented by the constants `MAX_VALUE` and `MIN_VALUE`. The smallest and largest `int` values are defined in the `Integer` class. These class constants can be accessed by using the name of the class in which the constant is defined, followed by the dot operator (`.`) and the name of the constant. Therefore, to access the largest and smallest `int` values, use `Integer.MAX_VALUE` and `Integer.MIN_VALUE`.

### Experiment 3.1

In this experiment you will investigate the storage of integer values in Java.

**Step 1.** Compile and execute the program `J03E01.java`, and record the values of the smallest and largest `int` values.

```
class J03E01
{
    public static void main(String[] args)
    {
        System.out.println(
            "In Java the smallest int value is " + Integer.MIN_VALUE);
        System.out.println(
            "In Java the largest int value is " + Integer.MAX_VALUE);
    }
}
```

---



---



---



---



---



---



---

**Step 2.** Besides `int`, Java has three other integer types, `long`, `short` and `byte`. In this step, let's investigate the maximum and minimum values of the primitive data type `byte`. Information about and methods involving type `byte` are stored in a class called `Byte`. Both the `Integer` and `Byte` classes are in the API package `java.lang`, which is automatically imported into every Java program. Modify the program to print the smallest and largest `byte` values. Record the results.

---

---

---

---

---

---

**Step 3.** Since integers are represented in two's complement notation, `MAX_VALUE` should be one less than some power of two,  $2^n - 1$ , and `MIN_VALUE` should be a power of 2,  $2^n$ , for some integer  $n$ . Explain why.

---

---

---

---

---

---

---

---

**Step 4.** For what value of  $n$  is  $2^n - 1$  equal to `Byte.MAX_VALUE` and  $-2^n$  equal to `Byte.MIN_VALUE`?

---

---

---

---

**Step 5.** Use the information collected in Steps 2 through 4 to determine the number of bits used to represent data of type `byte`. How many bytes are used? Explain your answers.

---

---

---

---

---

---

---

---

**Step 6.** Using the values obtained in Step 1 and the analysis completed in Steps 2 through 5, determine the number of bits and bytes that are used to represent data of type `int`.

---

---

---

---

---

---

---

---

---

---

### Experiment 3.2

In this experiment you will determine how your computer system responds to programs that try to produce integer values that exceed the limits of `INT_MIN` and `INT_MAX`.

**Step 1.** Execute the following program `J03E02.java`, and record the results.

```
class J03E02
{
    public static void main(String[] args)
    {
        int number = Integer.MAX_VALUE - 2;
        int count = 0;

        while(count < 5)
        {
            System.out.println(number);
            number++;
            count++;
        }
    }
}
```

---

---

---

---

---

---

---

---

---

---

**Step 2.** Briefly explain the results obtained in Step 1.

---

---

---

---

---

---

---

---

---

---

**Step 3.** Modify the program from Step 1 to record what happens on your system when you try to represent integers smaller than `Integer.MIN_VALUE`. Record the results.

---

---

---

---

---

---

---

---

**Step 4.** Explain the results obtained in Step 3.

---

---

---

---

---

---

---

---

---

---

---

---

## Data of Type Real

As with the storage of integers, only a finite number of real values can be represented in floating-point notation in a computer's memory. Other values must be rounded to a value that can be represented. Thus, a value stored as type `double` is often merely an approximation of the desired value. The class `Double` contains information and methods pertaining to the primitive type `double`.

### Experiment 3.3

**Step 1.** Execute the program J03E03.java and record the value of `Double.MIN_VALUE`.

```
class J03E03
{
    public static void main(String[] args)
    {
        System.out.println(
            "In Java the smallest double value is " + Double.MIN_VALUE);
    }
}
```

---

---

---

**Step 2.** Modify the program in Step 1 to print the value of the largest value that can be stored as a double. What changes did you make?

---

---

---

---

---

---

---

---

---

---

---

**Step 3.** Add the steps below (which assign a value with many significant digits to a double variable and then prints the variable) to the program in Step 1. Record the results.

```
double d = 987.65432109876543210987654321;
System.out.println(d);
```

---

---

---

---

---

---

---



**Step 2.** What would the program in Step 1 produce if all values were stored accurately?

---



---



---



---



---



---



---



---

**Step 3.** Explain the discrepancy between your answers in Steps 1 and 2.

---



---



---



---



---



---



---



---

## Data of Type Character

In Java, data items of type character are stored as bit patterns according to the Unicode encoding scheme. In the text *Computer Science: An Overview* you learned that ASCII is an eight-bit code whereas Unicode is a 16-bit code. Moreover, Appendix C of *Computer Science: An Overview* contains a partial listing of ASCII. In a sense, this appendix is also a partial listing of Unicode because the Unicode encoding of each symbol listed can be obtained by merely adding eight 0s to the left of the ASCII encoding. Thus, since the character *a* is represented as 01100001 in ASCII, it is represented as 0000000001100001 in Unicode.

The following program (J03E05) illustrates how we can identify the bit pattern used to represent a character. First note that if the variable *ch* (of type *char*) is assigned the value 'a', then the statement

```
System.out.print(ch);
```

prints the character *a*. We can, however, request that the bit pattern assigned to *ch* be interpreted as though the variable was of type *int* via the statement

```
System.out.println((int)ch);
```

Here the expression

```
(int)ch
```

is an example of casting. In general, a cast takes the form

*(newDataType) variable*

where *newDataType* is the data type to be applied to the bit pattern assigned to *variable*. Therefore, if the variable *ch* (of type *char*) is assigned the value 'a', then the statement

```
System.out.println((int)ch);
```

prints the integer 97, which is the value obtained when the bit pattern 0000000001100001 (the Unicode representation of the symbol *a*) is interpreted as a two's complement representation.

### Experiment 3.5

**Step 1.** Compile and execute J03E05.java. Record the results.

```
class J03E05
{
    public static void main(String[] args)
    {
        char ch = 'a';
        System.out.println(ch + " " + (int)ch);
        ch = '2';
        System.out.println(ch + " " + (int)ch);
        ch = 'Z';
        System.out.println(ch + " " + (int)ch);
    }
}
```

---



---



---



---



---

**Step 2.** Modify the program in Step 1 to find the Unicode representations for the characters ?, 0 and space. Summarize your work and findings below.

---



---



---



---



---



---



---



---



---



**Experiment 3.6**

**Step 1.** Compile the program J03E06.java. Record and explain the results.

```
class J03E06
{
    public static void main(String[] args)
    {
        int x;
        double d;

        x = 5;
        d = x;
        System.out.println("int " + x + "    double " + d);

        d = 2.95;
        x = d;
        System.out.println("int " + x + "    double " + d);
    }
}
```

---

---

---

---

---

---

---

---

---

---

**Step 2.** Correct the program in Step 1 by replacing the statement

```
x = d;
```

with

```
x = (int)d;
```

Record the results.

---

---

---

---

---



**Step 2.** Compile J03E07B.java. Record the error message.

```
class J03E07B
{
    public static void main(String[] args)
    {
        char c;
        c = 'A';
        while(c < 'F')
        {
            System.out.println((int)c + " " + c);
            c = c + 1;
        }
    }
}
```

---

---

---

---

---

---

---

---

**Step 3.** Correct the program in Step 2 by replacing the statement

```
c = c + 1;
```

with

```
c = (char)(c + 1);
```

Record the results, and compare these results to the results of the program J03E07A.java in Step 1.

---

---

---

---

---

---

---

---

---

---

---

---



## Optional Post-Laboratory Problems

- 3.1.** Suppose you want to write a program to manipulate integers lying outside the range of `Integer.MIN_VALUE` and `Integer.MAX_VALUE`? Java provides the primitive data type `long` for such an occasion. Although `long` has a greater range of representation compared to `int`, it too has its limitations. Find these values which are defined in the class `Long`. Do the same for the fourth integer type `short` which has a related class `Short`. Present the results in a neat format in which the min and max values are recorded for each of the four integer primitive types, listing them in order according to the number of bits used for storage.
- 3.2.** There is a second floating-point primitive, `float`, and a related class `Float`. Write a program that prints out the `MIN_VALUE` and `MAX_VALUE` for a `float`. How does the range of `float` values compare to the range of `double` values? Which data type uses the larger number of bits for storage?
- 3.3.** Write a program that receives a user-entered integer and prints the integer as a binary numeral. Use the method

```
public static String toBinaryString(int n)
```

from the `Integer` class that, when given an integer value returns a `String` containing the integer's binary representation.

For example,

```
Integer.toBinaryString(13)
```

returns "1101" and, if `x` equals 4, then

```
Integer.toBinaryString(x)
```

returns "100".

- 3.4.** Write a program that prints decimal numbers in the range 1 to 200 and their binary and hexadecimal equivalents. Hint: The `Integer` class also contains the method

```
public static String toHexString(int n)
```

Here is partial output:

1	1	1
2	10	2
3	11	3
4	100	4
	.	.
200	11001000	c8

- 3.5.** Write a program that prints a table of the printable ASCII subset of Unicode characters. The printable characters have values of 32 to 126.
- 3.6.** A computer system does not necessarily support the entire Unicode character set. The `Character` class contains constants and methods pertaining to the primitive type `char`. Write a program that prints the max and min values for type `char` both as type `char` and as type `int`. Note: If no character appears to be printed, that could mean that the character is a space or is non-printable. If, a "?" is printed, that could mean that the character is a question mark or is undefined on this system. Refer back to Experiment 3.5, Step 2 in which you

found the Unicode value of the space and question mark. Include a short interpretation of the results. Does this confirm that Unicode uses 2 bytes to store values of type `char`?

- 3.7.** Modify the program written in 3.5 above, to print the printable characters that are defined on your system with Unicode values greater than 126.
- 3.8.** Design an experiment to confirm that in Java items of type `char` are encoded in 16 bits (Unicode) rather than eight bits (ASCII).