Java Wrapper Classes (Outsource: 13-59 – 13-62)

For every primitive type in Java, there is a built-in object type called a wrapper class. Wrappers classes are classes that wrap up primitive values in classes that offer utility methods to manipulate the values. For example, the wrapper class for int is called Integer; for double it is called Double. Wrapper classes are useful for several reasons:

- Each wrapper class contains special values (like the minimum and maximum values for the type) and methods that are useful for converting between types. The methods of the wrapper classes are all static so you can use them without creating an instance of the matching wrapper class. Note that once a wrapper has a value assigned to it that value cannot be changed.
- You can instantiate wrapper classes and create objects that contain primitive values. In other words, you can wrap a primitive value up in an object, which is useful if you want to invoke a method that requires an object type.

Basic Type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Creating wrapper objects

The most straightforward way to create a wrapper object is to use its constructor:

```
Integer i = new Integer (17);          or   Integer i = 17;
Double d = new Double (3.14159);       or   Double d = 3.14159;
Character c = new Character ('b');     or   Character c = 'b';
```

Extracting the values

Java knows how to print wrapper objects, so the easiest way to extract a value is just to print the object or assign it to a primitive data type.

```
Integer i = new Integer (17);
Double d = new Double (3.14159);
System.out.println (i);
System.out.println (d);
```

Alternatively, you can use the toString method to convert the contents of the wrapper object to a String

```
String iStr = i.toString();
String dStr = d.toString();
```

Finally, if you just want to extract the primitive value from the object, there is an object method in each wrapper class that does the job:

```
int iValue = i.intValue();          or    int iValue = i;
double dValue = d.doubleValue();    or    double dValue = d;
```

There are also methods for converting wrapper classes into different primitive types.

Useful methods in the wrapper classes

Wrapper classes are useful places to put common methods we'd like to see associated with the basic data types.

Most of these methods are *class* or *static* methods - they are associated with the class rather than an instance of that class. To call them, you specify the name of the class and then add the dot (.) as usual followed by the method name and arguments.

Method Summary – Character class

static int	**getNumericValue**(char ch) Returns the int value that the specified character represents.
static boolean	**isDigit**(char ch) Determines if the specified character is a digit.
static boolean	**isLetter**(char ch) Determines if the specified character is a letter.
static boolean	**isLowerCase**(char ch) Determines if the specified character is a lowercase character.
static boolean	**isUpperCase**(char ch) Determines if the specified character is an uppercase character.
static boolean	**isWhitespace**(char ch) Determines if the specified character is white space according to Java.
static char	**toLowerCase**(char ch) Converts the character argument to lowercase.
static char	**toUpperCase**(char ch) Converts the character argument to uppercase.

Method Summary – Integer class

static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.
static int	**parseInt**(String s, int radix) Parses the string argument as a signed integer in the radix specified by the second argument.
static String	**toBinaryString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 2.
static String	**toHexString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 16.
static String	**toOctalString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 8.
static String	**toString**(int i) Returns a String object representing the specified integer.
static String	**toString**(int i, int radix) Returns a string representation of the first argument in the radix specified by the second argument.

Method Summary – Double class

static double	**parseDouble**(String s) Returns a new double initialized to the value represented by the specified String, as performed by the valueOf method of class Double.
static String	**toString**(double d) Returns a string representation of the double argument.

Splitting Strings Into Arrays

The String class has a **split(***String pattern***)** method that splits a string around matches of a given regular expression. For example:

```
String str = "Jersey Village High School";
String[] list = str.split("\\s+");
```

would split the String str into an array of Strings containing the values **{"Jersey", "Village", "High", "School"}.**

NOTE: In this pattern, **\\s** means whitespace and the **+** symbol means one or more. Therefore the expression **"\\s+"** would work just as well on the string **"Jersey Village High School"**.

Parsing A String With The Scanner Class

The **Scanner** class allows an application to break a String into tokens (substrings).

The set of delimiters (the characters that separate tokens) may be specified by calling the **useDelimiter(***String pattern***)** method (where pattern is a regular expression).

A token is returned by taking a substring of the String that was used to create the Scanner object.

To use the Scanner class to parse a String you must do three things.

- Include the following import statement: **import java.util.Scanner;**
- Instantiate a Scanner object:
  Scanner reader = new Scanner(*string***);**
- set the delimiter by calling **useDelimiter(***pattern***)**. If you do not call useDelimiter then by default the delimiter is whitespace.
- Use the **next(), nextInt() or nextDouble()** methods to extract either a primitive data type (int or double) or a substrings from the Scanner object.

The String **str** must already contain the text to be tokenized before the Scanner object is instantiated.

The following is one example of the use of the Scanner class:

```
Scanner reader = new Scanner("this/is/a/test");
reader.useDelimiter("/");     // Uses '/' as a delimiter rather than white space.
while (reader.hasNext())
{
    System.out.println(reader.next());
}
```

The program shown above displays the following output:

```
this
is
a
test
```

LAB 17 - ASSIGNMENT

Lab 17A - 60 points

OBJECTIVE

Every device connected to the public Internet is assigned a unique number known as an Internet Protocol (IP) address. An IP address consists of a series of 32 binary bits (ones and zeros). It is very difficult for humans to read a binary IP address. For this reason, the 32 bits are grouped into four 8-bit bytes called octets. An IP address, even in this grouped format, is hard for humans to read, write, and remember; therefore, each octet is presented as its decimal value, separated by a decimal point or period. This format is referred to as dotted-decimal notation. When a host is configured with an IP address, it is entered as a dotted decimal number, such as 192.168.1.5. Imagine if you had to enter the 32-bit binary equivalent of this: 11000000101010000000000100000101. If just one bit were mistyped, the address would be different and the host might not be able to communicate on the network.

WAP that reads a sequence of 32 bit binary numbers from a data file (**"lab19a.dat"**) and converts them into IP addresses.

FIELD SUMMARY

- **String[ ] binary** – an array of binary numbers in the form of strings.
- **int count –** a counter to keep track of the number of array elements read from the data file.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input` and `output`.
- **input** – declare a `Scanner` object and read a sequence of binary numbers from a data file using the `Scanner's nextLine` method. Convert the sequence of numbers into an array of individual numbers using the String's `split` method.
- **IPAddress** - **Arguments:** a string – an IP address in the form of a binary number. **Return:** a string. Calculate and return the IP address in its dotted decimal number form. This method should be declared as ***static***. **Hint:** Use substring to break the binary number into four parts then use the appropriate `Integer` wrapper class method to convert the binary numbers to their equivalent decimal values.
- **output** – Output all the binary numbers in the array **`binary`** and their equivalent IP address.

SAMPLE DATA FILE INPUT

01010111111110000111101001111010
01001000000101011101011010000000
11001111001011101000001100101011

SAMPLE OUTPUT

01010111111110000111101001111010 = 87.248.122.122
01001000000101011101011010000000 = 72.21.214.128
11001111001011101000001100101011 = 207.46.131.43

Lab 17B - 70 points

OBJECTIVE

WAP that reads a sequence of decimal numbers from a data file (**"lab17b.dat"**) and converts them into binary values ($base_2$). Each number in the data file will be separated by a forward slash.

FIELD SUMMARY

- **int[ ] decimal** – an array of decimal ($base_{10}$) number.
- **int count** – an accumulator to keep track of the number of elements in the array decimal.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input` and `output`.
- **input** – declare a `Scanner` object that reads from a data file. Set the delimiter to a **"/"** and read a sequence of decimal ($base_{10}$) numbers using the `Scanner's nextInt` method..
- **output** – Use a for loop and the appropriate `Integer` wrapper class method to convert the decimal numbers into binary numbers.

SAMPLE DATA FILE INPUT

2/12/25/18/36/254

SAMPLE OUTPUT

2 (Base 10) = 10 (Base 2)
12 (Base 10) = 1100 (Base 2)
25 (Base 10) = 11001 (Base 2)
18 (Base 10) = 10010 (Base 2)
36 (Base 10) = 100100 (Base 2)
254 (Base 10) = 11111110 (Base 2)

Lab 17C - 80 points

OBJECTIVE

WAP that converts decimal (base$_{10}$) numbers into binary, octal, and hexadecimal numbers. The program should present the user with a menu to select the conversion format and then should prompt the user to enter a number. Convert the number into the specified format. *This program should run three times.*

FIELD SUMMARY

- **int decimal** – a number entered from the keyboard in decimal (base$_{10}$) format.
- **char format –** a character specifying which number format conversion to process.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input` and `output`.
- **input** – declare a `Scanner` object and read a number format conversion type and a number to convert from the keyboard using appropriate prompts.
- **output** – Use a switch statement and the appropriate `Integer` wrapper class method to perform the appropriate number format conversion and output the results.

SAMPLE KEYBOARD INPUT

Number Systems
B - Decimal to Binary
O - Decimal to Octal
H - Decimal to Hexadecimal
Conversion Format: **h**

Enter the number to convert: **395**

SAMPLE OUTPUT

395 (Base 10) = 18b (Base 16)

Lab 17D - 90 points

OBJECTIVE

WAP that reads a sentence from the keyboard. For each character in the sentence, determine if the character is a letter, a number, a symbol or whitespace. Output each character and it attribute.

SAMPLE KEYBOARD INPUT

Enter a sentence: **Apollo 11, Moonwalk**

SAMPLE OUTPUT

'A' is a letter.
'p' is a letter.
'o' is a letter.
'l' is a letter.
'l' is a letter.
'o' is a letter.
' ' is white space.
'1' is a number.
'1' is a number.
',' is a symbol.
' ' is white space.
'M' is a letter.
'o' is a letter.
'o' is a letter.
'n' is a letter.
'w' is a letter.
'a' is a letter.
'l' is a letter.
'k' is a letter.

Lab 17E - 100 points

OBJECTIVE

WAP that translates an encrypted message read from a data file (**"lab17e.dat"**). The encryption scheme is quite simple. If a character is a lowercase letter add one to the characters value. Thus 'c' becomes 'd'. Of course, 'z' would become 'a'. If a character is an uppercase letter subtract one from the characters value. Thus 'C' becomes 'B' and, of course, 'A' would become 'Z'. White space, punctuation, and numbers remain unchanged.

SAMPLE DATA FILE INPUT

QqnHSzNlJOf hT Gtm!

SAMPLE OUTPUT

PROGRAMMING IS FUN!