Chapter 7
Useful Java Objects

We've already looked at many useful objects from the ACM Java library— GRects, GRectangles and so on. In this chapter we look at useful objects from standard Java and then in more detail at the RandomGenerator class from the ACM library.

7.1 the char Data Type and the Character class

Single characters such as we type at the keyboard are very useful. Standard Java provides the char data type and the Character class which provide all the power we need.

7.1.1 Working with Chars

The primitive data type char is suitable for storing single characters.

Declaring and assigning char values is easy.
char ch0; ← no value is assigned, but the memory location ch0 is created.
char ch1 = 'm'; ← chars are surrounded with single quotes, not double quotes
char ch2 = '□'; ← a space is a perfectly valid char
char ch3 = '+' ← you can assign any keyboard character to a char variable
char ch4 = ''; ← here's a double quote '
char ch5 = '\'; ← and this is how the single quote ' is represented. There's a backslash followed by a single quote, both inside a pair of single quotes.

There are couple of special characters that may be useful for output.

- 't' ← written as two keyboard characters, represents the <Tab> character, generally used to align output with tab marks (perhaps every 5 or 8 characters).
- 'n' ← represents the <Enter> character, and is used to move output to the next line
- both the <Tab> character and the <Enter> character are sometimes used as literals in output statements. The <Enter> character is known as the End-Of-Line character to software developers. Below is an example of output using these special characters, which we initially declare as constants.

```java
final char TAB = '\t';
final char EOL = '\n';
print("Hello" + TAB + "World!" + EOL);
print("Bye" + EOL + "bye");
```

```
Hello    World!
```
We can also include the <Tab> or <Enter> characters directly in the **String** literals that are output.

```java
print("Hello\tWorld\n");
print("Bye\nbye");
```

would produce exactly the same output as shown above.

**chars** can also be input from the keyboard, but we'll have to defer that discussion until we've looked at the **String** class.

### 7.1.2 The Character class

The standard **Character** class provides many methods for working with individual characters. We will consider just a few of these methods but there are many more. Each method is referenced using **Character.methodName**(argument).

- **Character.isDigit**(char) \(\Leftarrow\) return true if char represents a digit (0..9), false otherwise
- **Character.isLetter**(char) \(\Leftarrow\) return true if the char represents a letter (a..z or A..Z), false otherwise
- **Character.isLowerCase**(char) \(\Leftarrow\) return true if the char represents a lower case letter, false otherwise
- **Character.isUpperCase**(char) \(\Leftarrow\) return true if the char represents an upper case letter, false otherwise
- **Character.toLowerCase**(char) \(\Leftarrow\) if char is an upper case letter, return its lower case equivalent. Otherwise return a copy of the original character. Either way the original character is not changed
- **Character.toUpperCase**(char) \(\Leftarrow\) if char is a lower case letter, return its upper case equivalent. Otherwise return a copy of the original character. Either way the original character is not changed.

Here's a simple demo program.

```java
//TestChars.java
import acm.program.*;

public class TestChars extends ConsoleProgram
{
    public void run()
    {
        char c1 = 'a', c2 = '9';
    }
}
```
println(Character.isDigit(c1));
println(Character.isDigit(c2));
println(Character.isLetter(c1));
println(Character.isLetter(c2));
println(Character.toUpperCase(c1));
} //run
} //TestChars

And the output is
false
true
true
false
A

7.2 the String class

String is a standard Java class suitable for storing sequences of characters such as "hello world", "The code is ^&%" or "1 + 3 = 4". Strings are very useful for storing information such as names, addresses, social security numbers, test answers, and so on.

The Java String library is automatically included with every program we write, so we don't need to import anything. Note that a String is an object, unlike ints, doubles, booleans and chars, which are a primitive data types. The String class makes an extensive set of methods available to us.

7.2.1 String Variables

Just as we can have int, double, boolean and even GRect and GOval variables, we may also have String variables.

Declaring a String variable
- String phrase1, message;
- String inputValue;

These are String variables that have no assigned value.

Assigning a value to a String that has been declared
- phrase1 = "Now and in the future";
- message = phrase1 + "□we'll always love Java."; □ represents a blank

Recall that the + operator, when applied to Strings, performs concatenation. Thus message now has the value "Now and in the future we'll always love Java.".

Declaring and assigning a String
- String name = "Mork from Ork";
• \texttt{String s1 = ""; \leftarrow} creates an empty \texttt{String}, a perfectly valid \texttt{String} that has nothing in it. Note that this isn’t the same as
\texttt{String s1;}
which declares \texttt{s1} to be a \texttt{String} that has no value whatsoever.

You may have noted that our technique for declaring and assigning a value to a \texttt{String}
\[ \texttt{String address = "123 Elm Blvd";} \]

is significantly different from the technique for assigning a value to other objects. For example, a statement declaring and assigning a value to a \texttt{GOval} might be
\[ \texttt{GOval oval1 = new GOval(100, 50);} \]
The difference is because the designers of Java decided that \texttt{Strings} were so common that a shorter way of assigning a value would be convenient. However, we can also use the more conventional method for creating a \texttt{String} if we wish.
\[ \texttt{String address = new String("123 Elm Blvd");} \]
\[ \texttt{String anEmptyString = new String();} \]

7.2.2 Inputing Strings from the Keyboard
We’ve already seen \texttt{String} output using the \texttt{print()} or \texttt{println()} methods. It is also easy to input an entire line from the keyboard, storing it into a \texttt{String} variable. Here’s an example.
\[ \texttt{String code; code = readLine("Enter the secret code: ");} \]
\[ \texttt{print("The code is: " + code);} \]

A sample run of this code section might be
\begin{verbatim}
Enter the secret code: My name is Mickey Mouse
The code is: My name is Mickey Mouse
\end{verbatim}

7.2.3 Strings and chars
There are some additional things to know about \texttt{Strings} and \texttt{chars}.
• each character in the \texttt{String} is a \texttt{char}.
• we can \texttt{add} single characters to an existing \texttt{String} using the concatenation operator.
  \[ \texttt{char ch1 = '□';} \]
  \[ \texttt{char ch2 = '!';} \]
  \[ \texttt{String s1 = "hello" + ch1 + ch2; \leftarrow s1 is "hello□!";} \]
• note that the statement
  \[ \texttt{String s2 = ch1;} \]
is illegal. A char cannot be assigned to a String, even though it can be added to a String. So if we really need to put a single char into a String, we could use String s2 = "" + ch1; which is an inelegant solution but it works.

- a String may contain a single char, but it is still a String, as in String ms = "s";
- the positions of characters in a String are counted beginning at 0 and are called indexes. So the first character in a String has index 0, the second character has index 1, and so on. "hello" would be indexed as shown below.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
</tr>
</tbody>
</table>

This will be very important for some of the String methods below.

### 7.2.4 String Methods

We've seen how the ACM graphics objects such as GReets have quite a few methods such as move(), setColor() and so on. String objects also have many methods that you can use. Some of the most important are described below. Others can be found with a web search for 'java string'.

**Comparison methods**

Used to compare two string for order or equality.

**compareTo** - string1.compareTo(string2)

- compareTo returns a value less than 0 if string1 < string2 by values from the ASCII table (which is discussed later).
- compareTo returns 0 if the string1 is equal to string2.
- compareTo returns a value greater than 0 if string1 > string2.
- ordering is by character values from the ASCII table, which is fairly similar to dictionary ordering. Note however that upper case letters are less than lower case letters in the ASCII table, so DOG is less than Dog which is less than dog.
- an example
  ```java
  if (s1.compareTo(s2) == 0)
      println(s1 + " is alphabetically equal to " + s2);
  else if (s1.compareTo(s2) < 0)
      println(s1 + " is alphabetically less than " + s2);
  else
      println(s2 + " is alphabetically less than " + s1);
  ```
- Take a look at asciitable.com to see the ASCII values of the keyboard characters.

**compareToIgnoreCase** - compares two Strings as described above except ignores the distinction between upper and lower case.

**equals** - string1.equals(string2) - returns true if the Strings are exactly equal, including case, and false otherwise.
• an example
  if (s1.equals(s2) == true)
  {
    ...do something...
  }
else
  {
    ...do something else...
  }

equalsIgnoreCase – string1.equalsIgnoreCase(string2) – checks two Strings for equality without regard to upper and lower case differences.

Searching methods
Searching methods are used to determine the position of a char or String inside another String.

string1.indexOf(char1) - return the index of the first position at which char1 is found within string1; -1 is returned if the charValue is not found

string1.indexOf(char1, intFromIndex) - return the index of the first position at which char1 found within string1, starting the search at intFromIndex; return -1 if not found

string1.indexOf(string2) - return the index of the first position at which string2 is found within string1; -1 if not found

string1.indexOf(string2, intFromIndex) – return the index of the first position at which string2 is found within string1, starting the search at intFromIndex; -1 if not found

Some examples of these methods – for each, assume s1 is "Sam", s2 is "Samuel" and s3 is "Sammy Sammy Sammy!".

• s1.indexOf('m') ← return 2 because 'm' is at position 2 in "Sam"
• s1.indexOf('t') ← return -1 because 't' is not in "Sam"
• s3.indexOf('m', 7) ← return 8 because the first 'm' is found at position 8 if the search begins at position 7 of "Sammy Sammy Sammy!"
• s1.indexOf('S', 1) ← return -1 because 'S' is not in "Samuel" if the search begins at position 1
• s2.indexOf(s1) ← return 0 because "Sam" begins at position 0 in "Samuel"
• s3.indexOf(s1) ← return 0 because "Sam" begins at position 0 in "Sammy Sammy Sammy!". The search starts at position 0 unless an alternate starting position is specified
• s2.indexOf(s1, 3) ← return -1 because "Sam" is not in "Samuel" if the search begins at position 3
• s3.indexOf(s1, 9) ← return 12 because "Sam" begins at position 12 in "Sammy Sammy Sammy!" if the search starts at position 9

©Daniel L. Schuster, 2009, all rights reserved
What happens if we specify an invalid `intFromIndex`? For example the expression `s2.indexOf(s1, 19)` means begin the search at position 19 in a `String` that has only 18 characters. In this case the methods above return -1, indicating the search failed.

**Copying methods**

These methods return a copy of segments of the `String`:

- `string1.charAt(intIndex)` ← return a copy of the `char` at `intIndex` in `string1`
- `string1.substring(intBeginIndex)` ← return a copy of the `String` in `string1` that begins at `intBeginIndex`, using all the characters to the end of `string1`
- `string1.substring(intBeginIndex, intEndIndex)` ←return a copy of `String` in `string1` from position `intBeginIndex` to position `intEndIndex` – 1

If an invalid index is specified the program crashes, throwing a `StringIndexOutOfBoundsException` exception. We should be careful to code so that the indexes specify valid positions in the `String`.

**Miscellaneous methods**

- `string1.length()` - return a count of the characters within `string1`; note that due to the zero based indexing of the positions in the `String`, the `length()` is 1 more than the last index. For example, if the `length()` is 10 that means that the last character is at index 9.
- `string1.toLowerCase()` - return a copy of `string1`, in all lower case letters
- `string1.toUpperCase()` - return a copy of `string1`, in all upper case letters

7.2.5 An Example Program

Below is a program `NameNormalize1` that inputs a name in the form "firstname middlename lastname" or "firstname lastname" and outputs the result in a somewhat normalized form, with the middle name, if any, removed. Here are some examples.

- "George Herbert Bush"  →  "George Bush"
- "George Bush"  →  "George Bush"

This program uses the `String` methods `indexOf()` and `substring()` and `+` operator. Trace through this program with the above examples and the method descriptions to really understand how it works.

**NameNormalize**

```java
//NameNormalize1.java
import acm.program.*;

public class NameNormalize1 extends ConsoleProgram
```

©Daniel L. Schuster, 2009, all rights reserved
```java
{ 
    public void run()
    {
        final char BLANK = '□';

        String lname, name = readLine("Name please?");

        int firstBlankPos = name.indexOf(BLANK);
        String fname = name.substring(0, firstBlankPos);

        int lastBlankPos = name.indexOf(BLANK, firstBlankPos + 1);

        if (lastBlankPos != -1) //there was a middle name
            lname = name.substring(lastBlankPos + 1);
        else //no middle name
            lname = name.substring(firstBlankPos + 1);

        String resultName = fname + BLANK + lname;
        println(resultName);
    } //run
} //NameNormalize
```

### 7.2.6 Debugging with String Methods

The **String** methods themselves can be counted on to be bug free. If you have bugs in an application that appear to be related to **String** methods, then the problem is with the way you are using the methods.

Debugging with **String** methods generally requires two things:

- knowing what arguments you are supplying to the **String** method.
- understanding exactly what the **String** method does.

Therefore, we typically debug with **String** methods by:

- displaying the values of arguments before the method call in question.
- displaying returned values after the method call.
- carefully reading and understanding the **String** method description.

Here's an example. Suppose the value assigned to `word2` in

```java
word2 = word1.substring(n, x);
```

is not the expected value. We could debug this statement with

```java
println("*** N: " + n + " X: " + x + " WORD1: " + word1);
word2 = word1.substring(n, x);
```
println("*** word2: " + word2);

If the values of \( n \), \( x \) or \( word1 \) are not as expected, then we debug the code above the `substring()` call to see what is producing the unexpected values.

If \( n \), \( x \) and \( word1 \) are all as expected and yet \( word2 \) is not what was expected, then we must be using the `substring()` method incorrectly. Are the arguments \( (n \) and \( x \)) in the correct order? Do we understand the `substring()` method? Reread the method description carefully.

### 7.3 The Math Class

The Java Math class provides constants and methods that are useful for mathematical calculations. It is not necessary to `import` the Math library, it is automatically included.

The Math class is an example of what is called a static object, which is a Java object that can't be instantiated. In other words, you can't make a Math object. The program below will not compile.

```java
//MathObject
import acm.program.*;

public class MathObject extends ConsoleProgram
{
    public void run()
    {
        Math m = new Math();
        double x = m.sqrt(8);
        println(x);
    } //run
} //MathObject
```

The compilation error is `Math( ) has private access in java.lang.Math` which really means "You can't make a math object."

The Math class is an example of what might be called a tool class, that is a library that just provides tools, not objects.

#### 7.3.1 Constants in the Java Math library

The value of \( \pi \) (~3.14159) can be referenced with `Math.PI` and the approximate value of \( e \) (~2.71828) with `Math.E`. Here's an example that uses both.

```java
double m = Math.PI * Math.E;
```
7.3.2 Methods in the Java Math library

You can think of each method as a "button" on the calculator—plug in a number and get a number back. Below are descriptions of a few of the most useful Math methods.

Math.abs(numericValueX) – returns the absolute value of $x$ in the same data type as the argument.

Math.sqrt(numericValueX) – returns the $\sqrt{}$ as a double.

Math.cos(numericValueX), Math.sin(numericValueX), ... – returns the value of the trig functions as a double. $x$ must be in radians.

Math.pow(numericValueX, numericValueY) – returns $x^y$ as a double.

Math.min(numericValueX, numericValueY) – returns the minimum of $x$ and $y$, in the same data type as the arguments.

Math.max(numericValueX, numericValueY) – returns the maximum of $x$ and $y$ as a double.

Math.round(numericValueX) – return the nearest whole number to $x$, as an int, using standard rounding.

And here’s an example.

double one = Math.pow(Math.cos(2), 2) + Math.pow(Math.sin(2), 2);

7.3.3 Debugging with the Math Methods

You can rely on the Math methods to work correctly, if used correctly. Debugging consists of displaying the values of any arguments before the call and the returned result. Ask yourself the following:

- Are the arguments correct?
- Are they in the correct order in the method call?
- Do you understand the method? For example, the trigonometric functions require an argument in radians, not degrees—are you supplying radians?
- Are you assigning the method result correctly?

7.4 the ACM Random Class

Random numbers and other random values are extremely useful for simulations and game software. The ACM library provides a simple mechanism for generating several kinds of random values.
### 7.4.1 Random Generator, Random Ints and Seeding the Generator

We've already covered this, so a quick review will suffice.

```java
import acm.util.*; \(\Leftarrow\) bring in the random generating tools

RandomGenerator \(rg =\) new RandomGenerator( ); \(\Leftarrow\) mechanism for generating randoms

int \(\text{diceValue} = \textit{rg}.nextInt(1, 6); \) \(\Leftarrow\) random numbers from 1 to 5 inclusive

int \(\text{cardValue} = \textit{rg}.nextInt(-5, 20); \) \(\Leftarrow\) random number from -5 to 20 inclusive

\(\textit{rg}.setSeed(0); \) \(\Leftarrow\) seed the random number generator so that the randoms repeat with each run

Remember that you usually want to delete the \textit{setSeed}() statement when bugs are squashed.

The code segment below will generate the same sequence of random integers each time it runs.

```java
RandomGenerator \(\textit{ranGen} =\) new RandomGenerator();
\(\textit{ranGen}.setSeed(0);

for(int \(i=\)0; \(i<\)5; \(i++\))
{  println(\(\textit{ranGen}.nextInt(1,10));\)
} //generate 5 randoms
```

### 7.4.3 Creating Random Doubles

You can also use a \texttt{RandomGenerator} to create a random \texttt{double} in the range \textit{minimum value} \(\leq\) \textit{random} \(<\textit{maximum value}}. Note the less than sign. Mathematically this range is expressed \([\textit{min}, \textit{max})).

Examples:
\(\texttt{double \(\textit{randomTemp} = \textit{rg}.nextDouble(0, 5*\texttt{Math.PI}); \Leftarrow \textit{randomTemp} \text{ is assigned a random value from } [0, 5\pi).}\)

\(\texttt{double \(\textit{val} = \textit{rg}.nextDouble(.5, 10.3); \Leftarrow \textit{val} \text{ is assigned a random value from } [.5, 10.3)}\)

### 7.4.4 Creating Random Booleans

\texttt{true} and \texttt{false} values may be randomly generated with \texttt{randomGeneratorObject.nextInt}(), as in

\(\texttt{boolean \textit{freeThrowGood} = \textit{rg}.nextBoolean( );}\)
A random boolean value with a specific probability of having a true result may also be generated. For example, if you want `freeThrowGood` to be true 81% of the time, use

```java
boolean freeThrowGood = rg.nextBoolean(.81);
```

And here's an example use:

```java
if (freeThrowGood == true)
{  // code to execute when the free throw is good
}
else
{  // code to execute when the free throw is not good
}
```

### 7.4.5 Creating Random Colors

Random colors can be generated. Here's an example statement and a small program.

```java
Color c1 = rg.nextColor();
```

**RandomColorDemo**

```java
//RandomColorDemo.java
import acm.program.*;
import acm.graphics.*;
import acm.util.*;
import java.awt.*;
public class RandomColorDemo extends GraphicsProgram
{
    public static final int APPLICATION_WIDTH = 500;
    public static final int APPLICATION_HEIGHT = 300;
    public void run()
    {
        setBackground(Color.BLACK);
        GOval moon = new GOval(APPLICATION_WIDTH/2-25,
                                APPLICATION_HEIGHT/2-25, 50, 50);
        moon.setFilled(true);
        moon.setColor(Color.WHITE);
        add(moon);
        RandomGenerator rg = new RandomGenerator();
        while(true)
        {
            pause(500);
        }
    }
}
```
7.4.6 Creating Random Chars

There is no ACM or Java method for creating a random char. However, we can create a random int then convert it, via a type cast, to a char.

Recall that the integer codes for common English language characters and symbols are contained in the ASCII character table. It's printed in many programming books and you may find it online at www.asciitable.com.

Creating a random lower case character is easy.

```
char ch = (char) rg.nextInt(97, 122); ← lower case letters from 'a' to 'z'
```

This example program displays 10 random upper case characters.

```
RandomCharDemo
//RandomCharDemo.java
import acm.program.*;
import acm.util.*;

public class RandomCharDemo extends ConsoleProgram
{
    public void run() {
        RandomGenerator rg = new RandomGenerator();
        for(int i=0; i<10; i++) {
            println((char) rg.nextInt(65, 90));
        } //generate 10 random chars
    } //run
} //RandomCharDemo
```

7.4.7 Creating Random Strings

There is no ACM or Java method for creating a random String, but it's easy if we remember that a random String is nothing more than a sequence of random char values.

The example program below generates and prints a string of 10 random upper case letters in the range 'A' to 'Z'.

```
RandomStringDemo
```

"moon.setColor(rg.nextColor());
    } //endless loop
} //run
} //RandomColorDemo"
//RandomStringDemo.java
import acm.program.*;
import acm.util.*;

public class RandomStringDemo extends ConsoleProgram
{
    public void run()
    {
        RandomGenerator rg = new RandomGenerator();
        String rs = ""; //empty string
        for(int i=0; i<10; i++)
        {
            rs = rs + (char) rg.nextInt(65, 90); //add random chars to the string
        } //generate 10 random chars and add to the string
        println(rs);
    } //run
} //RandomStringDemo

Console Problem Set - Strings
For all problems, assume you are dealing with first and last names such as "George" or "Smith" and not "Mary Beth" or "van der Linn", unless otherwise stated. Also assume that there is exactly one blank between names, as in "Tom Sawyer" and not "Tom Sawyer", unless otherwise stated.

1. LastNameFirst Write a program that inputs a two part name such as "Andrea Anderson" and outputs it in the form "Anderson, Andrea".

2. InitialName Write a program that inputs a three part name such as "William Blythe Peters" and outputs it in the form "W. B. Peters"

3. FirstAndLast Write a program that inputs a two, three or four part name, such as "George Bush", "George Walker Bush" "George Herbert Walker Bush" and outputs it in the form "George Bush", "George W. Bush" or "George H. W. Bush", respectively

4. EncryptString Write a program that inputs a line of information. Replace the first and last 'A' (or 'a') with 'C', the first and last 'B' (or 'b') with 'P', the first and last 'O' (or 'o') with 'X'. Use find, rfind and []. For example, if the input was "It is a good day to go home." would become "It is C gXod dCy to go hXme.". Use a loop, not find or rfind.

5. ExtractSSN Write a program that inputs a social security number, using readLine, in the form XXX-XX-XXXX, builds a new string with the dashes removed, and then outputs the string.

6. ExtractPhone Write a program that inputs a phone number, using getline, in the form (XXX) XXX-XXXX (including blanks) builds a new string with all extra characters removed, and then outputs the string.

7. ManyInitials Write a program that inputs a two, three or four part name (such as "George Bush", "George Walker Bush", "George Herbert Walker Bush") and outputs it in the form "G. Bush", "G. W. Bush" or "G. H. W. Bush", respectively.
8. **CheckPasswords** Input 2 strings representing a password and confirmation of the password, as frequently used when changing a password. If the password and the confirmation are not the same, output the first position at which they are different. Repeat till the user chooses to quit.

9. **CheckLower** Input a string. If the string consists entirely of lower case characters ('a' → 'z') output "Good" else output "Bad".

10. **UpcaseString** Input a string. Output the same string, entirely in upper case letters.
    "Thomas" → "THOMAS", "123 oak St." → "123 OAK ST."

11. **Palindrome** A palindrome is a sequence of characters that reads exactly the same frontwards as backwards. For example, "dad" is a palindrome, as is "mom", "zzz", "Y" and "amanaplanacanalpanama" (a man a plan a canal panama). "now is the time" is not a palindrome, and neither is "nowan". Write a program that inputs a string, and tells the user if the string was a palindrome.

12. **StripExtraBlanks** For this problem, I'll use □ to represent blanks. Now suppose you input a string such as "□Now□is□the□time□□for□□□□all□good□programmers." Write a program to remove the extra blanks and output the result, such as "Now is the time for all good programmers."

13. **PigLatin** Pig Latin is a silly 'language' based on English, but translated using the following scheme.
    a) words that begin with a consonant – the consonant and all characters up to (but not including) the first vowel are removed and added as a suffix along with the string "ay": "great" → "eat-gray"
    b) words that start with a vowel – have the string "ay" added as a suffix: "and" → "and-ay"
    Write a program that inputs a string and displays its Pig Latin equivalent.
    Here’s a couple of examples:
    
    Enter sentence: You speak great Pig Latin.
    Enter Sentence: This is a hard project.
    Translation: is-thay is-ay a-ay ard-hay oject-pray.

    This is fairly difficult, by the way.

14. **TalkLikeAPirate**

**Console Problem Set – the Math library**

1. **SineValues** The sine function is a well known example of a periodic function—a function whose values repeat at regular interval. Demonstrate this by displaying the values of the sin of 0, Π/4, 2Π/4, 3Π/4, ...to 4Π.

2. **EighthRoot** The \( \sqrt[8]{x} \) can be calculated using the Math.pow() or the Math.sqrt() methods—

\[
\sqrt[8]{x} = x^{\frac{1}{8}} = (x^{\frac{1}{2}})^{\frac{1}{4}} = \sqrt{x^{\frac{1}{2}}} = \sqrt[4]{x^{\frac{1}{2}}} = \sqrt[4]{\sqrt[2]{x}} = \sqrt[2]{\sqrt[4]{x}}
\]

Write a program to calculate several 8th roots two ways (using the pow and sqrt methods) and compare the results. With each result, calculate result^8 and compare it with the
original number. Does there seem to be a difference in the accuracy of the **pow** method and the **sqrt** method used these ways?

3. **Logab** The logₐb (read the log of b base a) can be calculated, for base a, using

   The Java method for calculating the ln(x) is **Math.log(numericValueX)**. Write a program to input several a,b pairs and calculate and display logₐb. Check your answer by calculating a^{your answer}. It should equal b.

4. **SineSeries** The sin(x) can be approximate using the series

   Write a program to input x and then calculate and display its sine. How long should the loop generating the intermediate values run? How do you decide to stop it?

   Note that this series is fairly accurate for x values from -4 to 4, but becomes inaccurate quickly for values outside this range.

5. **SineSeries2** Repeat the sineSeries calculation above, but try to optimize the computations so each step involves as few operations as possible. For example, 7! = 7*6*5*4*3*2*1 but it also equals 7*6*5!, and the 5! was calculated in the previous step. The second method would involve far few computations than the first method.

6. **SineGraph** Repeat the sineValues exercise, but instead of displaying the sine values, display a simple graph of the value. You should be able to get something that looks like this for the range 0 to 2π..

   ![](image)

7. **BabylonRoot** The ancient Babylonians had a nice method of computing square roots that can be applied using only simple arithmetic operations. To find a rational approximation for
\[ \sqrt{N} \], let \( k \) be any number such that \( k^2 \) is less than \( N \). Then a new and better approximation for \( \sqrt{N} \), newK, is given by the formula

\[
\text{newK} = \frac{k + \frac{N}{k}}{2}
\]

Iterating this formula leads to \( k \) values that converge very rapidly to \( \sqrt{N} \).

For example, let's consider \( \sqrt{8} \). Let \( k = 2 \) (this is the first approximation). Then the 2\(^{\text{nd}}\) approximation, newK, is

\[
\frac{2 + \frac{8}{2}}{2}
\]

which is 3. Then third approximation, again called newK, is

\[
\frac{3 + \frac{8}{3}}{2}
\]

which is 2.83. The fourth approximation, newK, is

\[
\frac{2.83 + \frac{8}{2.83}}{2}
\]

which calculates to 2.828431. This is a very good approximation of \( \sqrt{8} \), accurate to less than .00001.

Write a program that inputs a value \( N \), a first guess for \( k \), and a desired accuracy \( \text{EPSILON} \). The loop repeats until \[ |\text{newK}^2 - N| < \text{EPSILON} \]. The last newk is output, and the user then has the opportunity to repeat the process for other values of \( N \).

**Graphics Problem Set**

1. **RandomTarget** Write a program that places a target in the approximate center of the application window. Play begins with a mouse click. The target moves in random jumps around the application window while the player attempts to click on it. If a jump takes the target off the application window then put the target back at the center. Display the number of clicks while the game plays. Play ends when the target is clicked.

2. **ShootTheUFO**