Chapter 1
Fundamentals

1.1 Getting Started in Console Programming

Back in the depths of time (like when I was in college) computer users interacted with a computer using only the keyboard and a simple screen. There were no graphics and no mouse, just white text on a black screen. We call this console mode, and it's where we'll start our work.

Standard Java works in console mode and is actually fairly simple to use. The ACM library provides a thin layer of tools that make program structure, screen output and keyboard input even simpler. We'll start there, with the basics.

1.1.1 the Hello World Program – Console Version

The traditional first computer program has long been a simple program that displays the phrase "Hello World" on screen. Writing and running this program requires a fundamental knowledge of Java basics as well as using your programming editor. Here's our version.

```java
public class HelloWorldConsole extends ConsoleProgram {
    public void run() {
        println("Hello World");
    }
}
```

Note that the line numbers are not part of the program. We've put them there just for easy reference.

Let's take a quick, line by line tour of this program:

1) This is a comment, which is text in the program used to provide information to humans reading the code. It is ignored by Java. This comment identifies the name of the file as HelloWorldConsole.java.

2) The import statement tells the program to include the basic ACM program library, making the code in that library available to our program.
3) This is just empty space, inserted for easier reading by humans.

4) The beginning of the program definition. Note that *HelloWorldConsole* is the name of the program and so the file name *must* be *HelloWorldConsole.java* Java program names always begin with a capital letter. *extends* *ConsoleProgram* means that this program will run in a simple console window, displaying only keyboard characters and getting input from the keyboard. No graphics, no mouse involved. This is the ACM Java version of the environment we old geezers used to work in.

5) The { marks the beginning of the real content of the program.

6) The **public void run** marks the beginning of the main part of the program. Things start to happen here.

7) The beginning of the content of the **run** method.

8) **println** is an output statement. It causes the phrase "*Hello world*", not including the double quotes, to be displayed in the console window.

9) The end of the content of the **run** method, with a comment to remind us of that.

10) The end of the content of the *HelloWorldConsole* program, with a comment.

Most of this is boilerplate, which means you'll put it in *every* console program you create. For simple programs only the comments, the name of the program (*HelloWorldConsole* in this case) and the content of the **run** section of the code will change.

We'll take a more in depth look at the basics and more shortly.

### 1.1.2 the Edit, Compile, Run cycle

A computer program is a set of instructions written in a programming language such as Java. The process of writing a program and being sure that it works requires several steps:

1) **edit** – entering the necessary computer code.

2) **compile** – submitting your program to the Java compiler, called compiling, allows it to checked for correctness. Unlike human languages, which are very flexible, programming languages are typically very inflexible. The compiler will check your code for correctness and provide error messages (which are not always helpful!) when there are errors. If there are no errors, then your program is translated into runnable form.

3) **run** - running the program allows you to view results and check its operation.

Programs are rarely perfect the first time, so you'll discover that you'll need to repeat this cycle many times for each program you write.

The edit, compile, run cycle and setup for two popular programming environments (TextPad and Dr. Java) are described in the Appendices.
1.2 Getting Started in Graphics Programming

Since the graphics computer revolution of the 80s and 90s, most computer users interact with a computer through the keyboard and the mouse, and the computer in turn displays detailed moving images. This is *graphics mode*, and we'll use it to do some interesting work.

Graphics programming in standard Java is difficult for the beginning student. Fortunately the ACM Java library provides a thick layer of tools that will help us with:

- creating simple and complex graphic objects
- creating behaviors such as movement
- modifying characteristics such as color
- reacting to the mouse
- and much more

We'll take a look at some of the ACM Library fundamentals later in this chapter, but for now let's start with another Hello World program.

1.2.1 the Hello World program – graphics version

Here we present the graphic version of the traditional first program.

```java
import acm.program.*;
import acm.graphics.*;

public class HelloWorldGraphics extends GraphicsProgram {
    public void run() {
        GLabel label1 = new GLabel("Hello world");
        add(label1, 100, 75);
    }
}
```

Let's look at the some of the lines in this program:

1) This statement loads the ACM graphics library, which makes a lot of easy to use graphics code available to us.

2) This is the beginning of the program itself. The *extends* `GraphicsProgram` means that this will run in a graphics windows. The program will therefore be able to do graphic output and mouse input. Hey, welcome to the 80s!

9) The *new* `GLabel` makes a graphic label for text. It is named `label1` and contains the phrase "Hello world".
10) The `add` method puts `label1` on screen, making it visible, at the location (100, 75) - more on the location later.

Most of this is boilerplate, which means you'll put it in every graphics program you create. Only the comments, the name of the program and the content of the `run` section of the code, will change.

### 1.2.2 A Quick Jump Into Animation

We won't look at it in depth, but below is the `HelloWorldGraphics` program, rewritten to demonstrate very simple animation. Note that the only changes are the addition of a `while` loop that causes the program to run forever, a `move()` method to move the `label` on screen, and a `pause()` statement to slow things down a little bit.

```java
// HelloWorldAnimation.java
import acm.program.*;
import acm.graphics.*;

public class HelloWorldAnimation extends GraphicsProgram {
    public void run() {
        GLabel label1 = new GLabel("Hello world");
        add(label1, 100, 75);

        while (true) {
            label1.move(1, 1);
            pause(100);
        } // move label
    } //run
}

//HelloWorldAnimation
```

When you are done with this course, you'll know quite a bit about basic animation.

### 1.2.3 the Edit, Compile, Run cycle

Assuming you have things set up correctly, the edit, compile, run cycle for graphic programs works exactly the same as for console programs.

### 1.3 Some Basic Java

Whether writing a standard Java program or an ACM console or graphics program, there's some basic knowledge we need to get started in programming. Some of these elements of the Java language were used in the above example programs.
1.3.1 Comments

A **comment** is text inserted into the programming code for the purposes of explanation to the people working on the code. Comments are ignored by the Java compiler and don’t become part of the running program.

Comments take a couple of forms:

- single line comments – a single line comment begins with `//` and continues to the end of the line
  - `//comment to the end of the line`
  - `println("Finished!"); //we're done!`
- multiple line comments – a multiple line comment begins with `/*` and continues until the first `*/`
  - `/* multi-line comment here, possibly over many lines, continuing to the */`
  - The `/*` and `*/` may also be used to create a single line comment, as in `/* this is a comment */`

Comments can also be used for removing sections of code from the compiled program. This is useful when debugging. For example, this statement

```
//println("The sum is: "+a+b);
```

is completely ignored because it is just a comment, and the last part of the statement

```
println("The sum is: "/a+b");
```

is ignored because it is enclosed in `/* */` marks.

1.3.2 import

Modern programming languages typically include dozens, perhaps hundreds, of prewritten modules of code called **libraries**. These libraries can be used in your program, so you don’t have to reinvent the wheel every time you write a program.

Java uses **import** to include libraries in the current program and that’s how we bring in some of the ACM libraries as well as various standard Java libraries to our program. **Import** statements are the first code after any comments, as you saw in the programs above.

Here are some examples.

```
import acm.program.*;
import acm.graphics.*;  //load the util library, which is part of java
import java.util.*;
```
1.3.3 White Space

Just as a book contains blank areas to separate sections of text (between paragraphs for example), so should a computer program. These blank areas are known as white space and are created with presses of the <SPACE>, <TAB> and <ENTER> keys.

Your code must have white space between components of the language, as in public void run instead of public void run, but Java itself ignores extra white space, with one exception we'll describe shortly.

Good use of white space dramatically improves the readability of a computer program. Note the difference between

```java
int x,y; double sumOfScores;println("Please enter the scores: ");
```

and

```java
int x, y;
double sumOfScores;
println("Please enter the scores: ");
```

Both have exactly the same statements and the same effect but the second group is clearly much easier to read and understand because of the white space.

1.3.4 Keywords, Standard and ACM Identifiers

Keywords are parts of the Java language such as public, class, int, while and so on. Keywords may not be used for any other purpose than their meaning as defined in Java.

Standard identifiers are words such as Scanner that are part of the libraries included with Java. You could create your own Scanner, overriding the standard meaning, but this is a very bad idea unless you are a real Java expert. We will treat standard identifiers the same as keywords and only use them as defined by Java.

ACM identifiers are parts of the ACM library such as GOval or waitForClick( ). They have meaning only for programs that import the needed ACM Libraries. We will use them a lot, particularly for our graphics program.

1.3.5 Some Primitive Numeric Data Types

Java stores simple information in what are called primitive data types. There are eight primitive data types. For us the most important numeric data types are:

- int – a data type suitable for storing whole numbers such as -84, 0 or 3274
  - Ints may range from -2 billion to +2 billion approximately.
• **ints** are stored exactly. If you put the int 32,457 into memory, then 32,457 is what is stored, without any loss of precision. It may surprise you but this is not true for all numeric information stored in computer memory.
• Calculations with **ints** are exact, without error. Except that if your program has integer overflow there’ll be a problem with accuracy, but we won't worry about it and you don’t have to know what integer overflow is for now.
• Calculations with **ints** usually run faster than calculations with **doubles**.
• Java automatically treats whole numbers in your source code as **ints**.
• In your source code you must put **ints** in your program without a comma. 32,457 would be typed in as 32457
• When running a program that asks you to enter an **int** at the keyboard you must type it in without a comma.

**double** – a data type suitable for storing real numbers such as -17.81 or 3.1415927, as well as whole numbers

• **doubles** may range from ±1.78x10^{308} to ±4.94x10^{-324}.
• **doubles** are not necessarily stored exactly. 2.980 might actually be stored as 2.9799999. This is a problem of course, but it won't generally concern us in a beginning Java course.
• **double** calculations are generally not exact and run slower than **int** calculations.
• in your source code, any number with a decimal point is automatically treated as a **double**.

Other Java numeric primitive numeric data types are **byte**, **short** and **long** (which are variations of the **int** data type) and **float** (which is a variation of the **double** data type). They won't be important for our first course in Java, so they won't be mentioned again.

There are also two other primitive data types, **boolean** and **char**, which will be very useful to us and we will study later.

Finally, let's consider how memory is allocated when we create variables for **primitive data types** and assign values to them. The statement

```java
int x;
```

results in a memory location big enough to hold an **int** with the name **x**. With reasonable (but certainly not perfect!) accuracy we can visual this as

```
x
```

empty memory location

The statement

```java
int x = -57;
```

gives us
Memory allocation for things like GLabels and other objects (we'll learn about them later) is more complex and we'll put that off for a while.

### 1.3.6 Identifiers

*Identifiers* are names for things that the programmer creates in the program — variables, constants, methods and classes for example. We've already used identifiers such as **HelloWorldConsole** and **label1**.

#### Rules for creating identifiers:
- Identifiers must consist of letters, digits and _ (underscores) *only*.
- Identifiers must begin with a letter. Technically this is not true. _ can be used to start an identifier also, but this is usually done only for advanced programming that we won't be doing, so we will consider this statement to be completely true.
- The number of characters in an identifier is theoretically unlimited but should be a reasonable limit. Something around 20 characters or less is usually about right.
- Identifiers can't be any of the Java keywords such as **public**. The program won't compile.
- Identifiers *shouldn't* be any of the Java standard identifiers such as **Scanner**, which causes programmer confusion though Java won't get confused. Java will use your meaning of the identifier, not the standard meaning.
- Identifiers *should*, but don't have to, be meaningful. **averageGPA** conveys much more meaning and is easier to read than **ag**.

You can use any capitalization technique for identifiers. Here are some examples:

- `maximumgammavalue`
- `maximum Gamma Value`
- `MaximumGammaValue`
- `maximumGammaValue`

A very common preference is **maximumGammaValue** or perhaps **maxGamma** — this capitalization style is called *camel caps* and it will be used in this book.

Here are example of good identifiers.

<table>
<thead>
<tr>
<th>total</th>
<th>sumOfScores</th>
<th>xPositionOnScreen</th>
</tr>
</thead>
<tbody>
<tr>
<td>body12</td>
<td>countOfClicks</td>
<td>maxScore</td>
</tr>
<tr>
<td>jobIsDone</td>
<td>middleInitial</td>
<td>verifyUserInput</td>
</tr>
</tbody>
</table>

Here are examples of bad identifiers that will nonetheless pass the Java compiler.

- `sumofscores` ← hard to read
x  \leftarrow \text{unless you are solving a mathematics problem, this name is meaningless}
maxBowlingScoreForTheLeagueDuringThe2010Season  \leftarrow \text{too long for reasonable typing}
sdfwljvls  \leftarrow \text{just plain stupid - meaningless, hard to remember, hard to type}

Finally we have examples of illegal identifiers which will be rejected when the program is compiled.

sum of scores  \leftarrow \text{blanks not allowed}
1rstScore  \leftarrow \text{can't begin with a digit}
f#  \leftarrow \text{can't use anything but letters, digits and _}
public  \leftarrow \text{public is a Java keyword, so you can't use it as an identifier}
william.darnell  \leftarrow \text{can't use anything but letters, digits and _}

1.3.7 Variables

A variable is a named memory location that can be assigned a value, and that assigned value can be changed during the running of the program. Variables must be declared using identifiers as described above. Here are some examples of variable declaration.

```java
int count;
int sum, alpha, beta;
double gpa, averageTestScore;
```

At this point, none of these variables have a value. This means that you can't use these variables yet except to assign a value to the variables.

A variable name should describe what is stored in a meaningful way. totalScores means a lot more than total, and t would be a terrible choice.

A default value is a value that is automatically assigned to a variable without the programmer writing code to perform the assignment. In Java sometimes there are default values, but often not. We will assume for now that there is never a default value, which means we'll always need to give initial values to a variable.

Assuming all of the variables below have already been declared, here are some assignment examples.

```java
count = 14;
alpha = -8;
averageTestScore = 17.4;
sum = count + alpha;
count = count + 10;
```

If a variable x is used before it is assigned a value the compiler will generally display an error.
int x;
println(x);

variable x might not have been initialized ← error message

1.3.8 Basic Math Operators

Java provides many useful math operators which we briefly describe.

- +, - and the unary negative (such as the negative in -7) all work as expected
- * is used for multiplication
- / - the division operator
  - Division is a bit more complicated because division with int is treated differently than division with doubles.
  - If x and y are both int, then x/y is the integer result of the division, with any fractional part thrown away. For example:
    - 10/8 has the value 1. The fractional part (2/8) is discarded.
    - 15/4 has the value 3. The fractional part (3/4) is discarded.
    - 100/20 has the value 5. There is no fractional part.
    - 7/12 has the value 0. The fractional part (7/12) is discarded.
  - If x or y or both of them are doubles, x/y is the result of performing real number division and the fractional part is retained. For example:
    - 9.3/4.2 has the value 2.2142 approximately.
    - 10/4.0 has the value 2.5 approximately.
    - 10.0/5 has the value 2.0 approximately.
    - Recall that double valued expressions are not guaranteed to be exact, which is why I wrote approximately.

Division with at least one double is calculated just as if both values were doubles.

- Maximum accuracy of int division can be forced by putting a double into the operation. For example, assume r is a double and x and y are ints. The correct result, including the fractional part, could be calculated and stored with
  
  \[ r = \frac{x \times 1.0}{y}; \]

  The 1.0 in the numerator forces x*1.0 to be evaluated as a double, and so the entire expression is evaluated as a double, thus maintaining accuracy. It’s an inelegant solution, but it works. Later we’ll learn a better solution called type casting.

- Division by 0 – you may recall that in mathematics division by zero is said to be undefined. A Java program that attempts to divide by 0 crashes (unless you’ve otherwise handled this possibility), producing an error message like this:

  Exception in thread "main" java.lang.ArithmeticException: / by zero

- % - the modulus operator
  - x % y - calculates the integer remainder when the int x is divided by the int y. If you don’t remember long hand division as done with paper and pencil, now would be a good time to review that topic, because the % operator gives you the remainder exactly as calculated by long hand division.
The % operator is generally used only with operands that are both positive, but does work correctly with negative operands.

- % examples:
  - 12 % 5 has the value 2
  - 10 % 5 has the value 0
  - 4 % 5 has the value 4
  - 43 % 43 has the value 0
  - -12 % 5 has the value -2
  - -12 % -5 has the value -2

- shortcut operators
  The operation of increasing or decreasing a value by 1 is extremely common. Incrementing x by 1 would be written

  \[ x = x + 1; \]

Java provides two useful shortcut operators for the increment and decrement operations.

<table>
<thead>
<tr>
<th>shortcut operator</th>
<th>example</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>x++;</td>
<td>( x = x + 1; )</td>
</tr>
<tr>
<td>--</td>
<td>x--;</td>
<td>( x = x - 1; )</td>
</tr>
</tbody>
</table>

This is two dashes, though it may look like one longer dash.

- order of operations – evaluation is left to right but modified by this order of operations, which should be familiar to you from arithmetic:
  1) unary negative (as in the minus sign in -13.4)
  2) *, / and %
  3) +, -

Examples:

<table>
<thead>
<tr>
<th>expression</th>
<th>evaluated as if written</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-x + y)</td>
<td>((-x) + y)</td>
</tr>
<tr>
<td>(x + a * b)</td>
<td>(x + (a * b))</td>
</tr>
<tr>
<td>(a * b + c * d)</td>
<td>((a * b) + (c * d))</td>
</tr>
<tr>
<td>(a * b % c - d * e)</td>
<td>(((a * b) % c) - (d * e))</td>
</tr>
</tbody>
</table>

We can use ( ) to group expressions. Note that ( ) is the only grouping symbol in Java. You can't use brackets ( [ ] ) or braces ( { } ) for grouping.

1.3.9 Constants

A **constant** is a named memory location whose value cannot change while the program is running.
Constants are named using identifiers as described above and identified by the keyword `final`.

```java
final int MAX = 100;
final double PIE = 3.14;
final int MAX_INPUT = 100;
```

It is common practice in Java, but *not a requirement*, to name constants with all upper case letters, as shown above.

### 1.3.10 String Literals

As you saw in the `HelloWorld` programs above, we often need to work with a phrase or set of characters. The phrase must enclosed in *double* quotes, as in "*Hello World*". Single quotes, as in ‘Hello World’, *will not work*. A phrase in double quotes is a *string literal*.

### 1.4 Fundamentals of ACM Java Console Programming

Console programs do keyboard input and text output and don’t pay attention to the mouse or display graphics. The ACM Java library includes methods that simplify creation of these programs. Let’s take a more in depth look at some essentials of console programming.

#### 1.4.1 ACM Console Program Template

Every ACM Java console program must include the following, which provides the basic functionality and structure for a console program.

```java
import acm.program.*;

public class TheNameOfTheClass extends ConsoleProgram {
    public void run() {
        ...executable code...
    }
} //TheNameOfTheClass
```

You will fill in `TheNameOfTheClass` and the executable code.

#### 1.4.2 Console Output

There are two ACM methods for displaying text information in the console window.

- `print()` – output and leave the output cursor on the same line
- `println()` – output and move the output cursor to the next line
Examples:

```java
print("Let's get started ");
println("right now!");
println("And what did you mean by that?");
```

which give us

```
Let's get started right now
And what did you mean by that?
```

White space in output is important for readability, but we must be careful. The lines below will not compile because the quoted `String` extends beyond the end of the line.

```java
print("Now is the time for all
good men to quit Java and have a beer.");
```

This will result in the error message `unclosed string literal` because we forgot the closing double quote mark " on the same program line as the opening double quote.

White space in a quote is not ignored. The statement

```java
println("It is a good day to die!");
```

displays with much more space between the words than

```java
println("It is a good day to die!");
```

1.4.3 The String + Operator

The `+` operator is used for arithmetic of course, but it can also be applied to `String` literals and `String` variables. When used in this way, `+` means 'concatenate', which means 'stick together'. In other words,

```java
print("hello" + "goodbye")
```

results in exactly the same output as

```java
print("hellogoodbye");
```

Both give us `hellogoodbye`

If you have a very long output line that does not fit in one source code line you may fix the problem in two ways:
• by using two output statements
  
  ```java
  print("Now is the time for all");
  print(" good men to quit Java and have a beer.");
  ```

• or by using the + concatenation operator
  
  ```java
  print("Now is the time for all"
  + " good men to quit Java and have a beer.");
  ```

You will often need to output a phrase followed by a variable. This is easily done using the `String +` operator. Here's an example:

```java
println(" The total is: " + total);
```

The phrase " The total is" is concatenated with the value of `total`. If `total` has the value 37 (for example) the result is

```
The total is: 37
```

However, the statement `println(" The total is: " + 10 + 20);` results in

```
The total is: 1020
```

Why? Well, the expression is evaluated as if we had

```java
println(" The total is: " + (10 + 20));
```

Since evaluating the expression in the inner most parenthesis gives us " The total is: 10", the next + concatenates the 20 onto this, giving us the final output above.

We can get the correct output with

```java
println(" The total is: " + (10 + 20));
```

using the ( ) to force evaluation of the arithmetic + before evaluation of the concatenation +.

### 1.4.4 Console Input

In standard Java keyboard input is a bit awkward. Fortunately the ACM library provides several useful input methods that make input easy. Here are several examples of numerical input.

```java
int value;
double startR;
value = readInt(" What is your score?"); ← prompt on screen
startR = readDouble(); ← no prompt on screen
```
A sample run of these two lines of code, with the user choosing to input 45 and then 38.4, might be

```
What is your score? 45
38.4
```

which puts 45 into `value` and 38.4 into `startR`.

Here's another example.

```java
double gpa = readDouble("Enter your gpa: ");
int firstValue = readInt("? ");
```

A sample run of these two lines of code, with the user inputting 3.4 and then 17, would look like this

```
Enter your gpa: 3.4
? 17
```

which would assign `gpa` the value 3.4 and `firstValue` the value 17.

A console input statements does the following:

1) halt the program
2) display a prompt if the prompt is included in the input statement
3) read the keyboard input when it is typed in and <ENTER> is pressed
4) return the value of the keyboard input
5) and then continue execution

Note that you can use a `readDouble()` to read an `int` from the keyboard, as in

```
Enter your gpa: 3
```

which will put the value 3.0 into `gpa`.

### 1.4.5 Console Program Example

This program inputs three test scores and outputs the correct average.

```java
public class ConsoleIODEmo extends ConsoleProgram
```

©Daniel L. Schuster, 2011, all rights reserved
{  
    public void run( )  
    {  
        int test1, test2, test3;  
        double ave;  
        test1 = readInt("Test 1? ");  
        test2 = readInt("Test 2? ");  
        test3 = readInt("Test 3? ");  
        ave = (test1 + test2 + test3)/3.0;  
        println("The average is: " + ave);  
    }  
}  
//run  
}  
//ConsoleIODemo  

1.5 Fundamentals of ACM Java Graphics Programming

The ACM Java library includes many tools that simplify creation of graphics programs that display simple and complex on screen objects. These programs can also work interactively with the mouse and the keyboard. Below is a very brief introduction to the ACM Graphics library.

1.5.1 Felt Board Metaphor

Graphics programming can be quite difficult, but when working with the ACM graphics library the familiar felt board from grade school is a pretty good metaphor to start with. Conceptually the operations are similar.

<table>
<thead>
<tr>
<th>Felt Board</th>
<th>Graphics Application Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. cut out a paper shape</td>
<td>1. create a graphic object</td>
</tr>
<tr>
<td>2. give some characteristics to the paper shape, perhaps by coloring it</td>
<td>2. give some characteristics to the object</td>
</tr>
<tr>
<td>3. add it to the felt board with a thumb tack</td>
<td>3. add the object to the graphics window</td>
</tr>
</tbody>
</table>

1.5.2 The Coordinate System

In algebra we use the familiar Cartesian coordinate system.

©Daniel L. Schuster, 2011, all rights reserved
We understand that though we draw points on paper and thus the drawn points have physical dimensions, conceptually the points are infinitely small.

In computer graphics we use a modification of this system as shown below.

![Diagram of coordinate system with points (0, 0) and (5, 2) on the X and Y axes.]

The origin of the graph is in the upper left corner of the screen. The points are called **pixels** and they definitely have physical dimension on the screen, albeit very small. Each pixel is a dot on the screen, and images are made by coloring these dots.

When we place an object on screen, the location of that object is ambiguous. For example, if we place a 30 pixel by 30 pixel rectangle as shown below, is the location of the rectangle at its center (25, 25) or at its upper left corner (10, 10)? Or perhaps it's somewhere else.

![Diagram of a 30x30 pixel rectangle with points (10, 10) and (25, 25) on the X and Y axes.]

Computer graphics systems typically decide that the location of a graphical object is the location of its upper left corner. So the rectangle above is said to be at (10, 10) even though it occupies many other pixels on screen.

### 1.5.3 The ACM Graphics Program Template

All ACM graphics program include the following code for basic structure and functionality.
import acm.program.*; //bring in the ACM graphics library
import acm.graphics.*; //identifies this as a graphics program

public class TheNameOfTheClass extends GraphicsProgram {
    public void run() {
        ...executable code here...
    } //run
} //TheNameOfTheClass

1.5.4 Graphics Objects

We have many different graphical objects available to us. Let's start with the two of them, rectangles and labels, and take a look at their essentials.

rectangles – rectangular objects are created with GRects. Here's an example.

GRect rect1 = new GRect(100, 70);

This statement creates an outlined black rectangle measuring 100 pixels wide by 70 pixels high, storing it in the GRect object named rect1.

add(rect1, 200, 300);

This statement puts rect1 on the screen, making it visible. rect1's upper-left corner is at (200, 300).

labels – are graphical objects containing keyboard characters such as letters and punctuation. They are typically used to display text messages to the user. Here's an example of creating a typical label.

GLabel label1 = new GLabel("Hello World!");

This statement creates a GLabel, naming it label1. Its content is the text phrase "Hello, world!". Our next step is to place this label on the screen so that it is visible.

add(label1, 40, 60);

This statement places the label on the screen at the point (40, 60). However, the location of a GLabel is not the location of its upper left corner. The location at which a label is displayed is the leftmost edge of the first character, along what is called the baseline, which is the line on which the uppercase letters sit. Some lowercase letters (g, j, p, q, and y) descend below the baseline, as do several special characters like the comma. The location of a GLabel and the concept of the baseline are illustrated in this diagram.
colors – the color of graphical objects may be set to different values. To do this we need

```java
import java.awt.*;
```

at the top of the program with the other import statements. This brings in the standard Java
Abstract Window Toolkit (AWT) library, which defines thirteen colors:

<table>
<thead>
<tr>
<th>common colors</th>
<th>other predefined colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color.BLACK</td>
<td>Color.PINK</td>
</tr>
<tr>
<td>Color.BLUE</td>
<td>Color.CYAN</td>
</tr>
<tr>
<td>Color.GREEN</td>
<td>Color.DARK_GRAY</td>
</tr>
<tr>
<td>Color.RED</td>
<td>Color.LIGHT_GRAY</td>
</tr>
<tr>
<td>Color.ORANGE</td>
<td>Color.MAGENTA</td>
</tr>
<tr>
<td>Color.WHITE</td>
<td></td>
</tr>
<tr>
<td>Color.YELLOW</td>
<td></td>
</tr>
<tr>
<td>Color.GRAY</td>
<td></td>
</tr>
</tbody>
</table>

Additionally you can use custom colors, which we'll cover later. The default color of any graphical object is Color.BLACK but it may be set with the setColor() method.

```java
label1.setColor(Color.BLUE);
rect1.setColor(Color.RED);
```

### 1.5.5 A Simple Graphics Example

Below is a simple program that puts a label on screen and displays two rectangles.

```java
//GraphicsDemo1.java
import acm.program.*;
import acm.graphics.*;
import java.awt.*;
public class GraphicsDemo1 extends GraphicsProgram
{
    public void run()
    {
        GLabel label1 = new GLabel("Graphics Demo #1");
        add(label1, 300, 250);
        GRect rect1 = new GRect(100, 50);
    }
}
```
add(rect1, 200, 300);
GRect rect2 = new GRect(30, 50);
rect1.setColor(Color.GREEN);
add(rect2, 500, 20);
} //run
} //GraphicsDemo1

1.5.6 The pause Statement

Doubtless you noticed that the labels and rectangles in the above demo program appeared simultaneously. You may cause a delay in statement execution, and thus a delay in the appearance of the objects, with the pause() statement.

pause(1000); ← pause for 1000 milliseconds, which is one second
pause(500); ← pause for ½ a second
pause(3500); ← pause for 3½ seconds

Here’s the GraphicsDemo1 program, modified with time delays thrown in.

GraphicsDemo2

//GraphicsDemo2.java
import acm.program.*;
import acm.graphics.*;
import java.awt.*;
public class GraphicsDemo2 extends GraphicsProgram
{
    public void run() {
        GLabel label1 = new GLabel("Graphics Demo #2");
        label1.setColor(Color.RED);
        add(label1, 300, 250);
        pause(250); //pause 1/4 of a second

        GRect rect1 = new GRect(100, 50);
        add(rect1, 200, 300);
        pause(3000); //pause 3 seconds

        GRect rect2 = new GRect(30, 50);
        rect1.setColor(Color.GREEN);
        add(rect2, 500, 20);
    } //run
} //GraphicsDemo2

The time used in pause() can be
  • an int literal value – pause(50);
- an int variable – \texttt{pause(x)};
- an int constant – \texttt{pause(WAIT1)};
- or even an int valued expression – \texttt{pause(2 * \textit{waitTime} + WAIT1)};

**Objective Questions**

**Objective questions**

1. TRUE/FALSE Java program file names must begin with a capital letter.
2. TRUE/FALSE Compiling checks your program for errors in your use of the Java language.
3. Describe the \texttt{import} statement.
4. Comments are
   a. ignored by the Java compiler
   b. translated into program code by the Java compiler
   c. used to write output statements
5. The code to begin a one line comment is: \\
6. // begins a comment that continues until
   a. the next // is encountered
   b. the end of the line
   c. the end of the program
   d. the next */ is encountered
7. White space consists of presses of what keys? \\
8. Extra white space is used to \\
9. Match each with appropriate description.
   
   | - keyword | - a part of the library that is provided to us by the ACM library |
   | - standard identifier | - part of the Java language that can't be used for anything else |
   | - ACM identifier | - part of the Java language that can be used for something else, but shouldn't be |

10. Circle the correct statements about \texttt{ints} and \texttt{doubles}
    a. calculations with \texttt{ints} run more slowly than calculations with \texttt{doubles}
    b. calculations with \texttt{ints} are exact; that is, they have no error
    c. \texttt{doubles} are stored exactly
    d. \texttt{ints} are stored exactly
    e. the range (difference between smallest and largest) of \texttt{ints} is greater than the range of \texttt{doubles}

11. __________ are names for things that a programmer creates in a program.
12. TRUE/FALSE Identifiers must consist of only letters and digits.
13. TRUE/FALSE \texttt{public} is a valid identifier.
14. Which are valid identifiers?
    a. _____ larryStorch
    b. _____ largest radius in model
    c. _____ hello!
    d. _____ GPA#
    e. _____ count_the_values
f. _____ intSumOfScores

g. _____ import

h. _____ println

i. _____ Averylongidentifierthatdoesntmeanmuchbutisitlegal

j. _____ tom'sBestGrade

15. Write a statement that declares and initializes an int variable sum to the value 0.

16. Evaluate each expression
   a. 2 + 6 * 5
   b. –3 + (5 * 3)
   c. 14 % 5
   d. 22 % 11
   e. 17 % 20
   f. 13 % 13
   g. 19 % 47
   h. 3 % 2
   i. 17 / 5
   j. 17 / 3
   k. 3/12
   l. 17/ 25
   m. 15/3
   n. 5*3 – 17 % 5
   o. 4*-5%7
   p. assume for each that x has the value 7 and evaluate the value of x after the expression is executed
      i. x+ +;
      ii. x- -;
   q. assume for each that x has the value -23 and evaluate the value of x after the expression is executed
      i. x- -;
      ii. x+ +;

17. TRUE/FALSE The statement final int MAX = 14.75; declares MAX to be a constant with the value of 14.75

18. TRUE/FALSE Assume MAX is declared as in problem 17. The statement MAX = 14; is a valid assignment statement.

19. Write a statement that declares E to be a constant with the value 2.71828

20. Write a statement that outputs the phrase "Now and again to the screen" in a console program.

21. TRUE/FALSE The statement print(" Total is: " + 1 + 2); outputs Total is 12.

22. Write statements to
   a. declare alpha to be an int and give it a value that comes from the keyboard, using an appropriate prompt.
   b. declare beta to be a double and assign it a keyboard value, using a prompt.
   c. declare ceta to be a double and assign it a keyboard value, without using a prompt.

23. In an ACM graphics window, the coordinate of the upper left corner is ________________

24. Fill in the missing code for a program that uses graphics.
public class PracticeProgram extends __________________________
25. Write statements to create a 50x25 graphics rectangle and add it to the graphics window at location (200, 210).
26. Write statements to create a label "Don't touch!" and put it in the window at (120, 120).
27. TRUE/FALSE Color.PURPLE and Color.LIME are not standard Java colors.
28. TRUE/FALSE Assume that robot is a rectangle on the screen. The statement to set robot to be blue is setColor(robot, Color.BLUE);
29. Write the statement to cause the program to wait for 5½ seconds.

**Short Answer Problems**

1. Write the statements to declare 2 integer variables, keyboard load them, and then display the result of dividing the first by the second, without losing any fractional part.
2. Assume block1 is a GRect that has already been declared and added to the screen. Write the statements to move block1 down and to the right by 5 pixels in each direction, wait for 10 seconds, then become red.
3. Assume x and y are integer variables, already loaded with values. Write the statements needed to calculate and display the result of dividing x by y, including the result and the remainder. For example, for 47/5 you would display:

   The result is: 9
   and the remainder is: 2

**Console Program Problems**

1) **SumAndProduct** Input three integers (stored as ints) from the keyboard, then calculate and display their sum and product.
2) **Address** Display your complete address in the Shire, the mythical land of Hobbits.
3) **Sphere** Input the radius of a sphere then calculate and display the volume. The formula you need is volume = 1.333 * Π * radius³. Use a constant for Π (~3.14159) and calculate radius³ with radius*radius*radius. Let radius be an int and volume be a double. If the radius input was 4, the display should look about like this

   Enter radius: 4
   My sphere has volume: 268.015 cubic inches.

4) **Box** Input the length, width and height of a box and then display this information along with the volume of the box. Let all variables be ints. Assuming the values 4, 10 and 3 were input, the screen should look like this

   Enter length: 4
   Enter width: 10
   Enter height: 3
   Length = 4, width = 10, height = 3 and volume = 120 cubic inches

5) **FtoC** The mathematical formula for converting a temperature measure in Fahrenheit degrees to Celsius degrees is: $C = \frac{5}{9}(F - 32)$. Write a program to input a Fahrenheit temperature as an int and output the result in Celsius degrees. Check your answer: $212^\circ F$
= 100° C, 32° F = 0° C. Quite possibly you will find you have an error in your result. Can you figure out why?

6) **ResistanceIsFutile** The formula for the resistance of 3 resistors in a parallel circuit is:

\[
R_{\text{circuit}} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}
\]

where \( R_1, R_2 \) and \( R_3 \) are the resistances of the individual resistors in the circuit. Write a program that inputs 3 ints representing resistances and calculates and displays the parallel circuit resistance. Do not allow truncation of fractional parts—*calculate the precise value.*

7) **ComputeChange** Inputs two ints. The first represents the amount of a purchase, in pennies. The second represents the funds tendered, in pennies. The program then computes and displays the appropriate change, in dollar bills, quarters, dimes, nickels and pennies.

Here is a sample run:

```
Enter amount of purchase (in pennies): 334
Enter amount tendered (in pennies): 500
Your change is:
1 dollar bill
2 quarter
1 dime
1 nickel
1 penny
```

You will probably need to use the % and / operators for calculations.

A couple of questions are in order, which you should consider, but it is not necessary to answer on paper:

a) Why was it important to input in pennies, as opposed to using 3.34 and 5.00?

b) Note "2 quarter" above. How would you make the output grammatically correct (1 quarter, 2 quarters), no matter what the number of quarters?

**Graphics Program Problems**

1) **FourSquare** Write a program to display 4 squares, size 30x30, of different colors, in various places on the screen. Put the label "FourSquare" at the bottom right corner of the graphics window, in blue.

2) **FourSquare2** Write a program similar to **FourSquare** but put the squares in the corners of the graphics window. The edge of each square should touch the edge of the window. You need some trial and error to figure out the size of the window, which you need to properly place the squares.

3) **Pyramid** Write a program to draw a simple pyramid, as shown here, with different colors for each layer.

```
©Daniel L. Schuster, 2011, all rights reserved  Page #24
```
4) **Grid** Write a program to draw this set of shapes as shown and display the label "Jumpin' Java", all in black.

![Grid diagram]

5) **Robot** Write a program to draw this crude robot in color, including the label. The elements of the robot should appear in the graphics window at intervals of ¼ second. Note how some rectangles overlap others—be sure to reproduce this characteristic.

![Robot diagram]