Variables, Data Types, Input & Math

Introduction

This unit covers how to gather data from the user and use that data to perform calculations.

Data

Literals

Literals are basic values that be interpreted in only one way. For example, 1 means one and nothing else.

Primitives

A **primitive** is a built in data type that holds a small amount of data (numbers, letters, booleans). All primitives start with lowercase letters. There are 8 primitive types; the four covered in this unit are in the following chart.

Type	Size	Description	Literal Example
Byte	1 byte	An integer value in range -128 to 127	124
Short	2 byte	An integer value in range -32,768 to 32,767	2300
Int	4 bytes	An integer value in range -2,147,483,648 to 2,147,483,647	56230
Long	8 bytes	An integer value in range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8596854645
Float	4 bytes	A floating point can be negative or positive and must be in the range 2^{-149} to $(2-2^{-23})\cdot 2^{127}$ with 7 digits of accuracy.	3.265541
Double	8 bytes	A floating point can be negative or positive and must be in the range 2^{-1074} to $(2-2^{-52})\cdot 2^{1023}$ with 17 digits of accuracy.	784.1236548975642
Char	2 byte	A single character	‘c’
Boolean	1 bit	A value that is either true or false	True

Objects

All other data in java is classified as an Object or class. **Objects / Classes** are complex data types that can store multiple pieces of data and perform actions. All Objects are created with special methods called **constructors**.

Variables

Variables are containers that hold data while your program is running.

Declaring a variable

The term used for creating a variable is **declaring**. Once a variable has been declared you never need to declare it again.

Format:

dataType variableName;

Example:

int age;

Declaring multiple variables at once

Format:

dataType variableName1, variableName2, …, variableNameX;

Example:

double radius, height;

Rules for choosing a variable name:

1. The variable name should start with a lowercase letter
2. The variable can only contain letters, numbers and underscores
3. A variable name can be of any length
4. Each word beyond the first should have uppercase first letter or be separated with an underscore.

Example variable names:

- studentName
- student_name
- student_Name

Assigning a value to a variable

The term used for giving a variable a value is called **assigning**. The first time a variable is assigned a value, it is called **initialization**.

Format:

variableName = value;

Example:

age =76;

Variable declaration and initialization in one line

A variable can be initialized when it is created. The format for doing this is as follows.

Format:

dataType variableName = value;

Example:

```
int studentNumber = 333333333;
```

Constant Variables

A **constant** is a variable that cannot be changed after it is initialized. When you make a constant variable it is convention to put its name in all capitals.

Format:

final type NAME = value;

Example:

```
final double PI = 3.14;
```

Math

Binary Math Operations

Binary operations are operations that involve two **operands**. An operand is a value used in a mathematical operation.

The result mathematic operation will be the data of the more complex operand. When a double is divided by an int the result will be a double. When an int value is divided by an int value the result is will be an int.

When working with only int values all decimal places get **truncated**, thrown away. For example 5/3 would be 1 and not 1.66.

Symbol	**Name**	**Description**	**Example**
+	Addition	Adds the value on the left to the value on the right.	5 + 8 Result – 13
-	Subtraction	Subtracts the value on the right from the value on the left.	22 - 5 Result – 17
*	Multiplication	Multiplies the value on the left by the value on the right.	4 * 7 Result – 28
/	Division	Divides the value on the left by the value on the right.	20 / 8 Result – 2 20.0 / 8.0 Result – 2.5
%	Modulus	Divides the value on the left by the value on the right. The result is the remainder of the division.	13 % 5 Result – 3

Assignment Operations

Assignment operations are binary operations that change the value of a variable. **Shortcut operators** are assignment operators that modify the current value of the variable instead of replacing it.

Symbol	**Name**	**Description**	**Example**

=	Equal / Assignment	Store the value on the right into the variable.	int x;
			x = 18;
			
			x is now 18.

+=	Plus Equals	Adds the right value to the variable.	int x = 12;
			x+=3;
			
			x is now 15

-=	Minus Equals	Subtracts the right value from the variable.	int y = 5;
			y-= 7;
			
			y is now -2.

*=	Multiple Equals	Multiplies the value of the right by the variable. The result is stored into the variable.	int x = 3;
			int y = 2;
			y*=x;
			
			y is now 6

/=	Divide Equals	Divides the variable by the value on the right. The result is stored into the variable.	double x = 5;
			double y =27;
			y/=x;
			
			y is now 5.4;

%=	Modulus Equals	Divides the variable by the value on the right. The remainder of the division is stored into the variable.	int x = 15;
			x%=4;
			
			x is now 3

Unary operations

Unary operations are operations that involve only one operand.

Symbol	Name	Description	Example

-	Sign Change	Uses the opposite sign of the value in the equation.	int x = 8;
			int y = -x;
			y would be -8 and x would still be 8.

+	Absolute Value	Uses the absolute value of the value in the equation.	int x = -3;
			int y = +x +5;
			y would be 8 and x would still be -3.

++(prefix)	Prefix Incrementation	Adds one to a variable and uses its new value in the equation.	int x = 12;
			int y = 3 - ++x;
			y would be -10 and x would be 13.

++(postfix)	Postfix Incrementation	Adds one to a variable, but uses its old value in the equation.	int x = 12;
			int y = 3 - x++;
			y would be -9 and x would be 13.

--(prefix)	Prefix Decrementation	Subtracts one from a variable and uses its new value in the equation.	int x = 3;
			int y = --x +2;
			y would be 4 and x would be 2.

--(postfix)	Postfix Decrementation	Subtracts one from a variable, but uses its old value in the equation.	int x = 3;
			int y = x-- +2;
			y would be 5 and x would be 2.

Operator Precedence

Operator precedence is the order in which operations are performed. A full precedence chart can be found here. For most circumstance the following 3 levels of precedence can be used:

Parentheses
Multiplication / Division / Modulus
Addition / Subtraction

When an equation is executed each level of precedence will be evaluated from left to right and then the next level will be processed.

Example:

(5+3) * 7 / 4 + 3 – (6 / 2)

Parentheses:

(5+3) * 7 / 4 + 3 – (6 / 2)
8 * 7 / 4 + 3 – (6 / 2)

Multiplication/Division:

8 * 7 / 4 + 3 – 3 **(After Parentheses)**
56 / 4 + 3 – 3

Addition/Subtraction:

14 + 3 – 3 **(After Multiplication/Division)**
17 – 3
14

Casting

Treating a value as a different type is called **casting**. Casting does not affect the value a variable stores.

Format:

(desired Type) value

Example:

6/4	=	1
(double)6/4	=	1.5
(double)(6/4)	=	1.0

Characters & Numbers

All chars have numeric values that can be accessed by casting them to an int. Any whole number can be cast as a char to get the character value it represents.

Import Character Values:

Char Value	Int Value
‘ ‘	32
‘A’	65
‘a’	97
‘0’	48

Examples:

```
int space = (int)(‘ ‘);
char letter1 = (char)(65);
char letter2 = (char)(99);
int number1 = (int)(‘H’);
int number2 = (int)(‘d’);
System.out.println(space);
System.out.println(letter1);
System.out.println(letter2);
System.out.println(number1);
System.out.println(number2);
```

Output:

```
32
A
c
72
100
```

String Class

Strings are Objects used to hold text. A String literal is text surrounded by double quotes.

Creating a String variable:

```
String name;
```

Assigning a value method 1:

```
name = new String(“text”);
```

Assigning a value method 2:

```
name = “text”;
```

Note: Method 2 is short a short-cut for method 1.

Concatenation

Concatenation is when a String value is joined to another piece of data. The join produces a String that contains text from both operands. The symbol used for concatenation is +.

Example of concatenation:

```
int x=5;
String s =“Bob likes the number”+x + “.“;
System.out.println(s);
```

Output:

```
Bob likes the number 5.
```

Scanner Class

The Scanner class was written to gather data from a data stream, usually the keyboard or a file.

Importing Scanner

The first step to using the Scanner class is to import the file containing the code for Scanner. **Importing** is letting your file know where to find other Object it will need. Imports are placed before your skeleton code.

Import for Scanner:

```
import java.util.Scanner;
```

Creating a Scanner

On the first line of main create a Scanner variable that stores a Scanner Object that reads data from the keyboard.

Format:

```
Scanner scannerVariableName = new Scanner(System.in);
```

Example:

```
Scanner keyboard = new Scanner(System.in);
```

Using a Scanner

Tell the user what you want them to enter. Next call the Scanner method that reads the type of data you want to gather and store the retrieved data into a variable.

Method	Description
nextByte()	Returns a byte value from the keyboard.
nextShort()	Returns a short value from the keyboard.
nextInt()	Gets an int value from the keyboard.
nextLong()	Returns a long value from the keyboard.
nextFloat()	Returns a float value from the keyboard.
nextDouble()	Returns a double value from the keyboard.
nextBoolean()	Returns a boolean value from the keyboard.
next()	Returns a single word value from the keyboard.
nextLine()	Returns multiple words from the keyboard.
next().charAt(0)	There is no method built into Scanner to get a single character from the keyboard, so this command is a workaround. This command gets a single word from the keyboard and then returns the first letter of the word.

Format:

```
variableName = scannerVariableName.scannerMethod();
```

Scanner Example:

```
Scanner keyboard = new Scanner(System.in);
int age;

System.out.print(“Enter your age: ”);
age = keyboard.nextInt();                  // user enters 8
System.out.println(“You are ” + age + “ years old.”);
```

Output:

```
You are 8 years old.
```

Math Class

The Math class has methods to perform complex operations and stores an accurate value of PI.

Accessing PI

Math stores an accurate value for PI.

Format:

```
Math.PI
```

Example:

```
area = radius*radius*Math.PI;
```

Math Methods

Method	Description
abs(int value)	Returns the absolute value of an int.
abs(double value)	Returns the absolute value of a double.
cos(double angle)	Returns a double value based on the cosine of the given angle value.
pow(double base, double exponent)	Returns a double value of the power.
random()	Returns a random double value from 0 to 1.
sin(double angle)	Returns a double value based on the sin of the given angle value.
sqrt(double value)	Returns a double that is the square root of the given value.
toDegrees(double radians)	Returns the degree value of the given value.
toRadians(double degrees)	Returns the radian value of the given value.

Accessing Mat Methods

Format:

```
Math.methodName(paramenters);
```

Example:

```
double power = Math.pow(2,8);
System.out.println("2^8 is " + power);
```

Output:

```
2^8 is 256
```

Printf

The printf command prints formatted text. The printf method takes multiple parameters. The first parameter is a control String that stores text and flags where data will be formatted and inserted into that text. The printf command leaves the cursor after the last printed value. For each flag there will be an additional parameter after the control String.

Adding a value to the control String

Percent signs are used to flag where data will be inserted into the control String. Following a percent will be 3 optional formatting settings and a type.

%	left align	minimum space	.decimal places	type

left align

- - left align
- Empty will be right aligned

minimum space

- # takes at least the number of given spaces
- Empty will take only the needed space

.decimal places

- .# rounds to the given number of decimal places
- Cannot be used on integer values

Type

- Sets the type of data to be inserted

Letter	**Type**
d	int
f	double
c	char
s	String

Percent Sign

To place a % inside the text you must use %%.

Printf Example

```
int age = 17;
double value = 9.8764534;
String name = “Jane”;

System.out.printf(“%s is %d years old.\n”,name,age);
System.out.println(“---%10.4f---\n”,value);
```

Output:

```
Jane is 17 years old.
---    9.8765---
```

Blue Pelican Sections

Lesson 3
Lesson 4
Lesson 5
Lesson 6
Lesson 7

Terms

Assigning	Giving a variable a value.
Assignment operations	Operators that change the value of a variable.
Binary operations	Operations that involve two operands.
Casting	Treating a value as another type.
Classes / Objects	Complex data types that can store multiple pieces of data and perform actions.
Constructors	Special methods used for the creation of an Object.
Declare	To create a variable.
Importing	Giving a file the location of a needed class.
Initializing	The first time a variable is assigned a value.
Literals	Values that can only be interpreted in only one way.
Method	A small segment of code designed to perform a simple task and possibly return a single value.
Operand	A value used in a mathematical operation.
Operator precedence	The order in which operations are performed.
Primitives	Built in data types that hold a small amount of data.
Shortcut operators	Assignment statements that are shortcuts to perform variable changes.
Truncation	When the data past the decimal point is dropped to get an integer value.
Unary operations	Operations that involve only one operand.
Variables	Containers that hold data while your program is running.

Full Precedence Chart

Operators	Description	Evaluation
[] . () ++ --	access array element access object member invoke a method post-increment post-decrement	Left to Right
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	Right to Left
() new	cast object creation	Right to Left
* / %	multiply divide modulus	Left to Right
<< >> >>>	shifts	Left to Right
< > <= >=	comparisons	Left to Right
== !=	equality	Left to Right
&	bitwise AND	Left to Right
^	bitwise XOR	Left to Right
|	bitwise OR	Left to Right
&&	conditional AND	Left to Right
||	conditional OR	Left to Right
?:	conditional	Right to Left
= += -= *= /= %= &= ^= != <<= >>= >>>=	assignment	Right to Left