Chapter 2, Data + Expressions Slides

Character Strings:

- a string of characters can be represented as a **string literal** by putting double quotes around it

ex:

"This is a string literal"

"123 main street"

"x"

- every character string is an object in Java, defined by the **String** class
- every string literal represents a **String** object

The println Method:

- we invoke the **println** method to print a character string
- the **System.out** object represents a destination (the monitor) to which we can send output

ex invoking a method:

```
System.out.println("Hello");
```

object — method name — parameter(s)

The print Method:

- **println** method is different from print method

example

- **println** method will print like

```
line 1
line 2
line 3
```

but!

- **print** method will print like `line 1 line 2 line 3`

String Concatination:

- the symbol use to concatinate strings is the + and appends one string to another

example: "Peanut butter" + "Jelly"

- it can also append a number to a string
- a string literal cannot be broken across two lines

- the + opperator can also be used for addition
  ↳ the function it performs depends on the type of info it's operating on

Escape Sequences:

- begins with the back slash (\)
- allows you to use quotes in a string literal so you don't break the code

example:

```
System.out.println("I said \"Hello\" to you.");
```

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

Variables:

- a variable is a name for a location of memory
  ↳ must be declared by specifying its name and info type it will hold

ex:

```
int total;
int count, temp, result;
```

data type → int; variable name → total

multiple variables can be created in one declaration

- you can initialize a variable with or without a value

ex: `int total = 50;`

type → int; variable name → total; optional optimization → = 50

- when a variable is used in a program, its current value is used

Assignment:

- assignment statements change the value of a variable
- the = sign is the assignment operator

example: `total = 55;`

- the expression on the right is evaluated and result stored in the variable on the right
- you can only assign a value to a variable that is consistant with the variables declared type

example: Assignment Statement:

1.) expression is evaluated
2.) result is assigned to variable

variable

```
height = height + gap;
```

assignment operator

Constants:

- a constant is an identifyer that is similar to a variable except it holds the same value throughout it's entire existance
  - ↳ as the name implies, it is constant, not variable

- compiler gives error if you try to change a constants value

- the final modifier is used to declare a constant

example: `final int MIN_HEIGHT = 69;`

- Constants are useful for three important reasons:

1.) they give meaning to othwise unclear literal values

2.) they facilitate program matenence
  - ↳ if a constant is used in multiple places, its value only needs to be updated in one place

3.) They formally establish that a value should not change, avoiding inadvertant errors by other programmers

Primative Data Types:

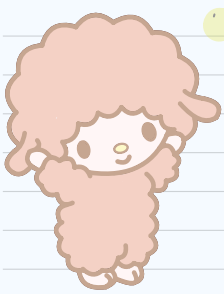
- There are eight primative data types in Java
  - ↳ four represent integers:
    - byte
    - short
    - int
    - long
  - ↳ two represent floating point numbers:
    - float
    - double
  - ↳ one represents characters
    - char
  - ↳ one represents boolean values (True or false)
    - boolean
- the difference between the various numeric data types is their size, and therefore the values they can hold

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	32 bits	Approximately -3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
double	64 bits	Approximately -1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Characters:

- char variables store a single character
- character literals are delimited by single quotes:

'a' 'x' '7' '$' ',' '\n'



example declarations:

```
Char topGrade = 'A';
char terminator = ';', separator = ' ';
```

Character Sets:

- character sets are an ordered list of characters
  - ↳ each character corisponds to a unique number

- a `char` variable can store any character from the `Unicode character set`
  - ↳ each character in the unicode character set uses 16 bits per character

- this is an international charater set
  - ↳ contains many symbols and characters from many world languages

Characters:

- the `ASCII character set` is old and smaller than Unicode
  - ↳ are a subset of unicode character set
  - includes:
    - uppercase letters
    - lowercase letters
    - punctuation
    - digits
    - special symbols
    - control characters

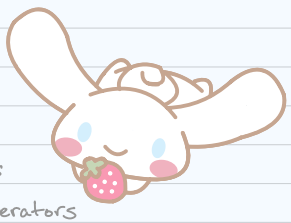
Booleans:

- `booleans` represent a true or false condition

- true or false reserve words and are the only valid values

- can also represent any two states → like a light bulb on or off

Expressions:



- an expression is a combination of one or more operators and operands
- arithmatic expressions compute numeric results and use the arithmetic operators

example of Arithmatic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- if either or both operands used by an arithmatic operator are floating point, then the result is a floating point

Division and Remainder:

- if both operands to the division operator (/) are integers the result is an integer
  ↳ the fractional part is discarded

example: 14 / 3 equals 4
8 / 12 equals 0

- the remainder operator (%) returns the remainder after dividing the second operand into the first

example: 14 % 3 equals 4
8 % 12 equals 8

Operator Precedence:

- operators can be combined into complex expressions

example:

```
result = total + count / max - offset;
```

- operators have precedence (order) which says how they are evaluated

example:
1) Parenthesis
2) multiplication / divison — L → R
3) addition / subtraction
4) string concatenation

- Parethisis can be used to force order of evaluation

More Assignment:

- the right and left hand sides of an assignment statement can contain the same variable

example:

first, one is added to the original value of count

```
Count = count + 1;
```

Then the result is stored back into count (overwriting the original value)

Increment and Decrement Operators:

- the increment and decriment operators use only one operand
- the **increment operator (++)** adds one to its operand
- the **decriment operator (--)** subtracts one from its operand

example:

this → `count++` is functionally equal to `count = count + 1`

- increment and decrement operators can be applied in:

example: Postfix form

↳ `count ++`

Prefix form

↳ `++ count`

- the two forms have different effects when used as a part of a larger expression
  ↳ because of their subtleties, incr and dec should be used w/ care

Assignment Operators:

- Java provides **assignment operators** to simplify the process of performing an operation on a variable, and then storing that result back in that variable

example:

```
num += count;
```

↓

is equivalent to

```
num = num + count;
```

- there are many assignment operators in Java

example:

operator	example	equivalent to
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

- the right hand side of an assignment operator can be a complex expression
- the entire right-hand expression is evaluated first, then the result is combined with the first original variable
  - ↳ therefore:

```
result /= (total-MIN) % num;
```

is equivalent to:

```
result = result / ((total-MIN) % num);
```

- the behaviors of some assignment operators depends on the types of operands
- if the operands to the += operands are strings, the assignment operator does string concatenation
- the behavior of an assignment operator (+=) is always consistant with the behavior of the corresponding operator (+)

Data Conversion:

- we sometimes have to convert data types

example: we might want an int as a float

- these conversions don't change the type of a variable or the value that's stored in it
  - ↳ they only convert a value as part of a computation

example:

```
int age = 42;
double someVariable = age * 0.5 // age remains an int when used age somewhere else
```

- conversions must be handled carefully to avoid loosing info

- widening conversions are safest because they tend to go from a small data type and go to a larger one

  example: like a short to an int

- narrowing conversions can lose info because they go from large data type to small

- in Java, data conversions can occur in three ways
  1) assignment conversion
  2) promotion
  3) casting

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Assignment Conversion:

- assignment conversion happens when a value of one type is assigned to a variable of another

- if `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`

example:

```
money = dollars
```

- only widening conversions can happen via assignment

- note that the value or type of `dollars` did not change

Promotion:

- promotion happens automatically when operators in expressions convert their operands

- for example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:

```
result = sum / count;
```

Casting:

- casting is the most powerful, and dangerous, technique for conversion

- both widening and narrowing conversions can be accomplished by explicitly casting a value

- to cast, the type is put in parenthese in front of the value being converted

- for example, if total and count are ints, but we want a floating point result when dividing them, we can cast total

  ↳ `result = (float) total / count ;`

The Scanner Class:

- the scanner class provides convenient methods for reading input values of various types

- a scanner object can be set up to read input from various sources, including the user typing values on a keyboard

- Keyboard input is represented by the System.in object

Reading Input:

- the following line creates a Scanner object that reads from the keyboard

  example:

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

- the new operator creates the scanner object

  - once created, the Scanner object can be used to invoke various methods, such as

    ↳ `answer = scan.nextLine();`

  - The Scanner class is part of the Java.util class library, and must be imported into a program to be used

- the `nextLine` method reads all of the input until the end of the line is found

* Some methods of the `Scanner` class

```
Scanner (InputStream source)
Scanner (File source)
Scanner (String source)
        Constructors: sets up the new scanner to scan values from the specified source.

String next()
        Returns the next input token as a character string.

String nextLine()
        Returns all input remaining on the current line as a character string.

boolean nextBoolean()
byte nextByte()
double nextDouble()
float nextFloat()
int nextInt()
long nextLong()
short nextShort()
        Returns the next input token as the indicated type. Throws
        InputMismatchException if the next token is inconsistent with the type.

boolean hasNext()
        Returns true if the scanner has another token in its input.

Scanner useDelimiter (String pattern)
Scanner useDelimiter (Pattern pattern)
        Sets the scanner's delimiting pattern.

Pattern delimiter()
        Returns the pattern the scanner is currently using to match delimiters.

String findInLine (String pattern)
String findInLine (Pattern pattern)
        Attempts to find the next occurrence of the specified pattern, ignoring delimiters.
```

Input Tokens:

- unless specified otherwise, `white space` is used to seperate elements (called tokens) of the input
- white space includes space characters, tabs, new line characters
- the `next` method of the `Scanner` class reads the next input token and returns a string
- methods such as `nextInt` and `nextDouble` read data of particular types