Unit 1: Primitive Types

Variables and Datatypes

Adapted from:

1) Building Java Programs: A Back to Basics Approach

by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum

https://longbaonguyen.github.io

Data Types

A **type** is a set of values (e.g. integers, floats, etc..) and a set of operations (e.g. +, -, *, /, etc..) on them.

Data types can be categorized as either **primitive** or **reference**.

The primitive data types used in this course define the set of operations for numbers and Boolean(true or false) values.

Reference variables or object variables hold a reference(or address) to an object of a class(more on this in a future lecture).

Primitive types

The primitive types on the Advanced Placement Computer Science A exam are:

- **int** - which store integers (whole numbers like 3, -76, 20393)
- **double** - which store floating point numbers (decimal numbers like 6.3, -0.9, and 60293.93032)
- **boolean** - which store Boolean values (either true or false).

Receipt example

What's bad about the following code?

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements
- We will use **variables** to solve the above problems.

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration

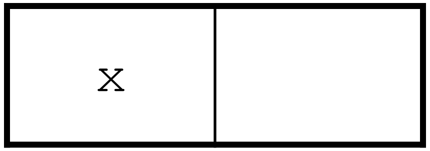
- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  type name;

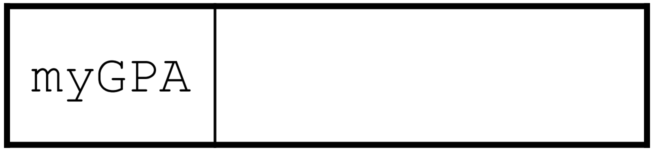
  - The name is an *identifier*.

  - `int x;`



x

  - `double myGPA;`



myGPA

Assignment

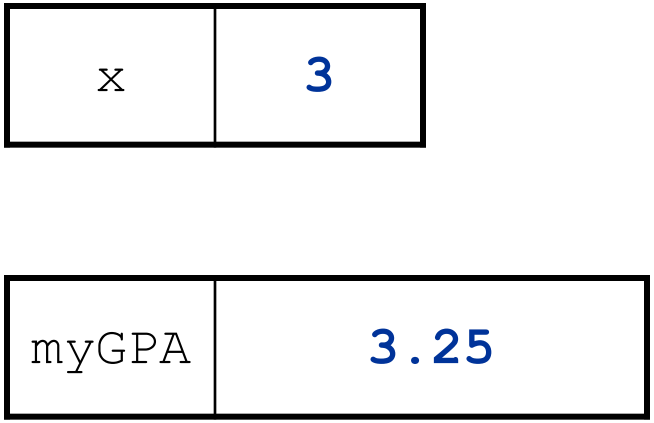
- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**;

  -
    ```
    int x;
    x = 3;
    ```

  -
    ```
    double myGPA;
    myGPA = 1.0 + 2.25;
    ```



x 3

myGPA 3.25

Using variables

- Once given a value, a variable can be used in expressions:

string concatenation:
string + number = concatenated string
(more on this later)

```
int x;
x = 3;
System.out.println("x is " + x);       // x is 3

System.out.println(5 * x - 1);         // 14
```

- You can assign a value more than once:

x	11

```
int x;
x = 3;
System.out.println(x + " here");       // 3 here

x = 4 + 7;
System.out.println("now x is " + x);  // now x is 11
```

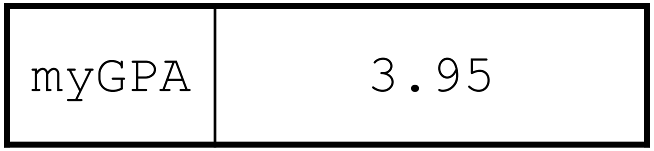
Declaration/initialization

- A variable can be declared/initialized in one statement.

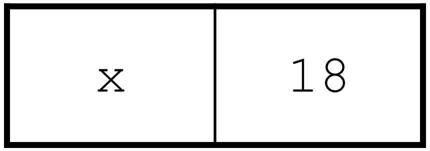
- Syntax:

  type name = **value**;

  - `double myGPA = 3.95;`



  - `int x = (12 - 3) * 2;`



Assignment and algebra

- Assignment uses `=` , but it is not an algebraic equation.

  `=` means, *"store the value at right in variable at left"*

  - The right side expression is evaluated first,
    and then its result is stored in the variable at left.

- What happens here?

```
int x = 3;
x = x + 2;   // no solutions
             // mathematically
             // not an equation!
```

x	5

Multiple Variables

- Multiple variables of the same type can be declared and initialized at the same time.

- Syntax:

type name1, name 2, name3;

```
int x, y, z;  // declare three integers.
```

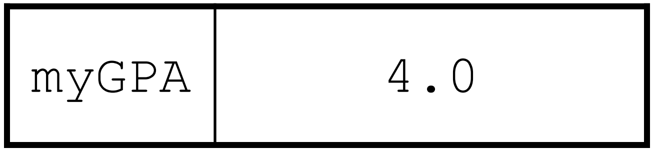
type name1 = value1, name2 = value2, name3 = value3;

```
int a = 1, b = 2, c = 3; // declare and initialize
                         // three integers.
```

Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;      // ERROR: incompatible types`

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`



myGPA	4.0

Compiler errors

- Order matters.

  -
    ```
    int x;
    7 = x;  // ERROR: should be x = 7;
    ```
- A variable can't be used until it is assigned a value.

  -
    ```
    int x;
    System.out.println(x);    // ERROR: x has no value
    ```
- You may not declare the same variable twice.

  -
    ```
    int x;
    int x;                    // ERROR: x already exists
    ```

  -
    ```
    int x = 3;
    int x = 5;                // ERROR: x already exists
    ```
    - How can this code be fixed?

Printing a variable's value

- Use + to print a string and a variable's value on one line.

  -
    ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 11 + 17 + 4 + 19 + 14;
    System.out.println("There are " + students +
                       " students in the course.");
    ```

- Output:

  ```
  Your grade was 83.2
  There are 65 students in the course.
  ```

Receipt question

Improve the receipt program using variables.

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);

        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);

        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);

        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .15 +
                           (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```

Type `boolean`

- **`boolean`**: A logical type whose values are `true` and `false`.

```
int age = 22;
boolean minor     = (age < 21);
boolean lovesAPCS = true;
System.out.println(minor); // false
System.out.println(lovesAPCS); // true
System.out.println(4 <= 5); // true
```

final

- The keyword **final** can be used in front of a variable declaration to make it a constant that cannot be changed. Constants are traditionally capitalized.

```
public class TestFinal
{
   public static void main(String[] args)
   {
      final double PI = 3.14;
      System.out.println(PI);
      PI = 4.2; // This will cause a syntax error
   }
}
```

Naming variables

The name of the variable should describe the data it holds. A name like `score` helps make your code easier to read.

A name like x is not a good variable name in programming, because it gives no clues as to what kind of data it holds.

Do not name your variables crazy things like `thisIsAReallyLongName`, especially on the AP exam. You want to make your code easy to understand, not harder.

Naming variables

The convention in Java and many programming languages is to always start a variable name with a lower case letter and then uppercase the first letter of each additional word.

Variable names **can not include spaces** so uppercasing the first letter of each additional word makes it easier to read the name. Uppercasing the first letter of each additional word is called **camel case**.

```
int numOfLives = 3; // camel case to highlight words
```

Another option is to use underscore symbol _ to separate words, but you cannot have spaces in a variable name. Java is case sensitive
so playerScore and playerscore are not the same.

```
int num_of_lives = 3;  // use _ to highlight words.
```

Keywords

- **keyword**: An identifier that you cannot use to name a variable because it already has a reserved meaning in Java.

abstract
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
extends
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
try
void
volatile
while

Input and `System.in` `(not on AP)`

- **interactive program**: Reads input from the console.
  - While the program runs, it asks the user to type input.
  - The input typed by the user is stored in variables in the code.

  - Can be tricky; users are unpredictable and misbehave.
  - But interactive programs have more interesting behavior.

- **`Scanner`**: An object that can read input from many sources.
  - Communicates with `System.in` (the opposite of `System.out`)
  - Can also read from files, web sites, databases, ...

Scanner syntax (not on AP)

- The Scanner class is found in the java.util package.

```
import java.util.*;    // so you can use Scanner
```

- Constructing a Scanner object to read console input:

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

  - Example:

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

Scanner methods (not on AP)

Method	Description
`nextInt()`	reads an `int` from the user and returns it
`nextDouble()`	reads a `double` from the user
`next()`	reads a one-word `String` from the user
`nextLine()`	reads a one-*line* `String` from the user

- Each method waits until the user presses Enter.
- The value typed by the user is returned.

```
System.out.print("How old are you? ");  // prompt
int age = console.nextInt();
System.out.println("You typed " + age);
```

- **prompt**: A message telling the user what input to type.

Scanner example (not on AP)

```
import java.util.*;    // so that I can use Scanner

public class UserInputExample {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("How old are you? ");
        int age = console.nextInt();


        int years = 65 - age;
        System.out.println(years + " years to retirement!");
    }
}
```

age 29

years 36

- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```

Input tokens (not on AP)

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by *whitespace* (spaces, tabs, new lines).
  - How many tokens appear on the following line of input?

```
23   John Smith    42.0  "Hello world"  $2.50   "   19"
```

- When a token is not the type you ask for, it crashes.

```
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output:

```
What is your age? Timmy
java.util.InputMismatchException
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        ...
```

Scanner example 2 (not on AP)

```
import java.util.*;    // so that I can use Scanner

public class ScannerMultiply {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();

        int product = num1 * num2;
        System.out.println("The product is " + product);
    }
}
```

- Valid Outputs (user input underlined):

```
Please type two numbers: 8 6
The product is 48

// 2 tokens separated by space
```

```
Please type two numbers: 8
6
The product is 48
// 2 tokens separated by new
// line
```

Lab 1: String Concatenation

Create a new repl on replit.

Create two integer variables: pens and pencils. Initialize it to some positive values. Use print statements and String concatenation to have the following output:

Output: (Assuming that the initialized values for pens=18, pencils = 10)

```
There are 18 pens.
There are 10 pencils.
Total is 28 pens and pencils.
```

Lab 2: String Concatenation

Use the SAME repl as from Lab 1 String Concatenation.

Modify your previous lab 1 to ask the user to enter the number pens and pencils rather than hard coding the values. Use print statements and String concatenation to have the following output:

Output:

```
Enter the number of pens: 10
Enter the number of pencils: 18
Total is 28 pens and pencils.
```

References

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum: https://runestone.academy/runestone/books/published/csawesome/index.html

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

https://longbaonguyen.github.io