Java Review

Marty Stepp Stanford Computer Science Dept.

علیرضا سازگار دانشگاه فردوسی مشهد

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.

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

- Syntax:

  type name = value;

  - `double myGPA = 3.95;`

  - `int x = (11 % 3) + 12;`

x	14

myGPA	3.95

متغیر ها

متغیر : بخشی از حافظه کامپیوتر است . که میتوان آن را مانند یک ظرف برای نگهداری داده ها در نظر گرفت.

متغیر ها را میتوان در یک خط هم تعریف کرد و هم به آن مقدار اولیه داد.

در صورت عدم مقدار دهی اولیه چه اتفاقی میافتد؟

Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

Name	Description	Examples
`int`	integers	`42, -3, 0, 926394`
`double`	real numbers	`3.1, -0.25, 9.4e3`
`char`	single text characters	`'a', 'X', '?', '\n'`
`boolean`	logical values	`true, false`

- Why does Java distinguish integers vs. real numbers?

انواع داده اولیه در جاوا

همانطور که اشاره شد هشت نوع داده اولیه وجود دارد.
int, char, byte, short, long, float, double, boolean

انواع داده بالا را از نظر حافظه مقایسه میکنیم:

byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

- Syntax:

  (**type**) **expression**

  Examples:

```
double result = (double) 19 / 5;      // 3.8
int result2 = (int) result;           // 3
int x = (int) Math.pow(10, 3);        // 1000
```

تغییر نوع داده

تغییر از نوع Widening:

```
byte -> short -> char -> int -> long -> float -> double
```

تغییر از نوع Narrowing:

```
double -> float -> long -> int -> char -> short -> byte
```

```
public class Main { public static void main(String[] args)
{
 int myInt = 9; double myDouble = myInt; // ?
 System.out.println(myInt); // Outputs 9
System.out.println(myDouble); // Outputs 9.0 }
 }
```

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable++;	**variable = variable + 1;**
variable--;	**variable = variable - 1;**

```
int x = 2;
x++;                        // x = x + 1;
                            // x now stores 3

double gpa = 2.5;
gpa--;                      // gpa = gpa - 1;
                            // gpa now stores 1.5
```

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `*` `/` `%` have a higher level of precedence than `+` `-`

    `1 + 3 * 4` is `13`

    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12      is 18
    ```

  - Parentheses can force a certain order of evaluation:

    `(1 + 3) * 4` is `16`

  - Spacing does not affect order of evaluation

    `1+3 * 4-2` is `11`

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42  is "hello42"
1 + "abc" + 2 is "1abc2"
"abc" + 1 + 2 is "abc12"
1 + 2 + "abc" is "3abc"
"abc" + 9 * 3 is "abc27"
"1" + 1       is "11"
4 - 1 + "abc" is "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
    - Output: `Grade: 83.5`

الحاق رشته ها

استفاده از concat:

```
s1.concat(s2);
```

استفاده از StringBuilder:

```
StringBuilder stringBuilder = new StringBuilder(100);
stringBuilder.append(“Java");
stringBuilder.append(" is");
stringBuilder.append(" awesome");
```

استفاده از join:

```
String[] strings = {"I'm", "running", "out", "of", "pangrams!"};
String myString = String.join(" ", strings);
```

Variable scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the `{ }` braces
    - A variable declared in a `for` loop exists only in that loop.
    - A variable declared in a method exists only in that method.

```
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    // i no longer exists here
} // x ceases to exist here
```

i's scope

x's scope

Class constants

- **class constant**: A value visible to the whole program.
  - value can only be set at declaration
  - value can't be changed while the program is running

- Syntax:

```
public static final type name = value;
```

  - name is usually in ALL_UPPER_CASE

  - Examples:

```
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 3.5;
public static final int SSN = 658234569;
```

Passing parameters

- Declaration:

```
public void name (type name, …, type name) {
    statement(s);
}
```

- Call:

```
methodName (value, value, …, value);
```

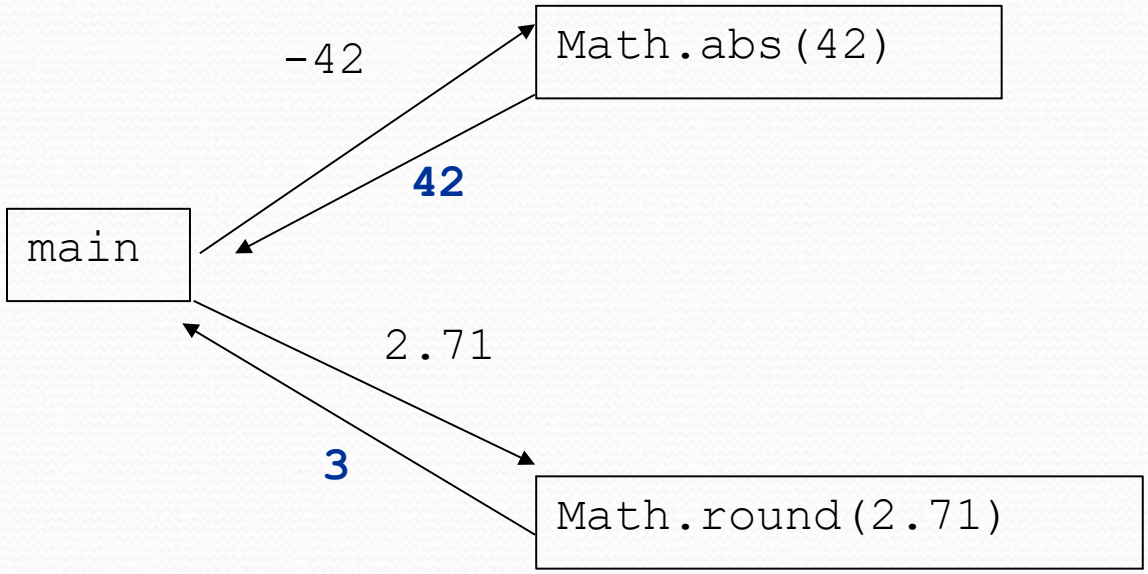
- Example:

```
public static void main(String[] args) {
    sayPassword(42);          // The password is: 42
    sayPassword(12345);       // The password is: 12345
}

public static void sayPassword(int code) {
    System.out.println("The password is: " + code);
}
```

Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.



Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.round(`*value*`)`	nearest whole number
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back
`Math.random()`	random `double` between 0 and 1

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Returning a value

```
public type name(parameters) {
    statements;
    ...
    return expression;
}
```

- Example:

```
// Returns the slope of the line between the given points.
public double slope(int x1, int y1, int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx;
}
```

Strings

- **string**: An object storing a sequence of text characters.

```
String name = "text";
String name = expression;
```

  - Characters of a string are numbered with 0-based *indexes*:

```
String name = "P. Diddy";
```

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	dy

  - The first character's index is always 0
  - The last character's index is 1 less than the string's length
  - The individual characters are values of type `char`

`String` methods

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1, index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());    // 7
```

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
    System.out.println("I need your TPS reports.");
}
```

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

Type `char`

- `char` : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);                    // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");   // P Diddy
```

char vs. String

- "h" is a String
  'h' is a char (the two behave differently)

- String is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();          // 'H'
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- char is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();
```

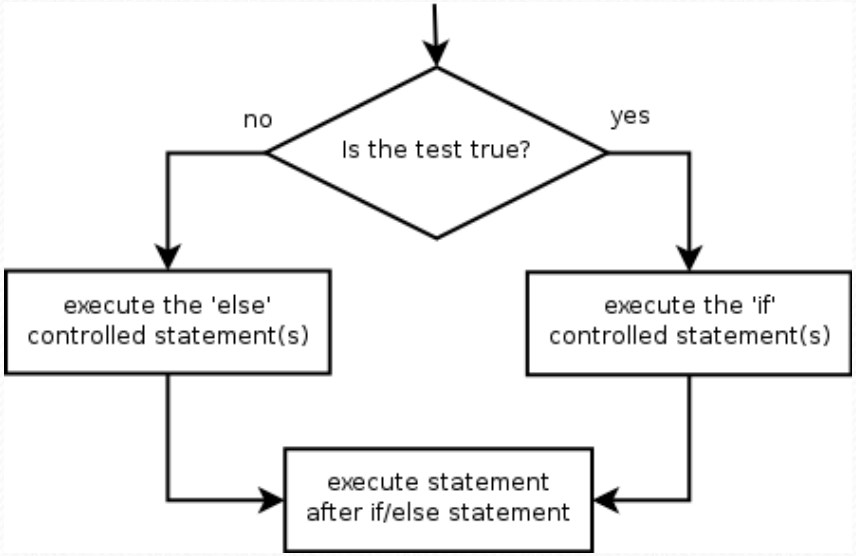
```
// ERROR: "cannot be dereferenced"
```

What is s + 1 ? What is c + 1 ?
What is s + s ? What is c + c ?

if/else

Executes one block if a test is true, another if false

```
if (test) {
    statement(s);
} else {
    statement(s);
}
```



- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to Mars University!");
} else {
    System.out.println("Application denied.");
}
```

Relational expressions

- A **test** in an `if` is the same as in a `for` loop.

```
for (int i = 1; i <= 10; i++) { ...
if (i <= 10) { ...
```

  - These are `boolean` expressions.

- Tests use *relational operators*:

Operator	Meaning	Example	Value
==	equals	`1 + 1 == 2`	`true`
!=	does not equal	`3.2 != 2.5`	`true`
<	less than	`10 < 5`	`false`
>	greater than	`10 > 5`	`true`
<=	less than or equal to	`126 <= 100`	`false`
>=	greater than or equal to	`5.0 >= 5.0`	`true`

Logical operators: &&, ||, !

- Conditions can be combined using *logical operators*:

Operator	Description	Example	Result
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`
`!`	not	`!(2 == 3)`	`true`

- "Truth tables" for each, used with logical values *p* and *q*:

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true