FCDS

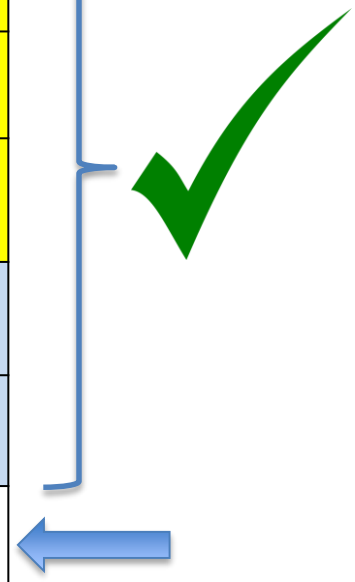
Programming I

Lecture 3: char, String object, Math class, Scanner

Java's primitive types

- **primitive types**: there are 8 simple types for numbers, text, etc.

Type	Description	Size
`int`	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range -128 . . . 127	1 byte
`short`	The short integer type, with range -32768 . . . 32767	2 bytes
`long`	The long integer type, with range -9,223,372,036,854,775,808 . . . -9,223,372,036,854,775,807	8 bytes
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 bit



- ***Java*** also has ***object*** types **(e.g. Strings)**, which we'll talk about later

Type `char`

char data type

- **char** : A primitive data type representing single characters of text (e.g., 'a', 'b', '@', ' ', etc.).

```
public static void main(String[] args) {
        char a = 's';
        System.out.println ("student" + a);
}
```

Output:

students

`char` vs. `int`

- Each **`char`** is mapped to an integer value internally
  - Called an **ASCII value**

`'A'` is 65	`'B'` is 66	`' '` is 32
`'a'` is 97	`'b'` is 98	`'*'` is 42

  - Mixing `char` and `int` causes automatic conversion to `int`.
    `'a' + 10` is 107, `'A' + 'A'` is 130
  - To convert an `int` into the equivalent `char`, type-cast it.
    `(char) ('a' + 2)` is `'c'`

Example

```
public static void main(String[] args)
{
	int x = 1;
	char letter1 = 'a';
	char letter2 = (char) (letter1 + 4);

	System.out.println(letter2);

	x = 'a' + 3;
	System.out.println(x);
}
```

Be Careful:

char x = 67; //correct since it takes the value as the ASCII code 67

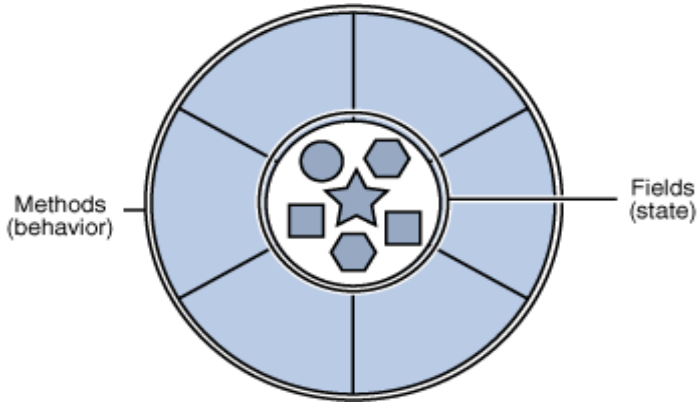
int y = 67;
char x = y; // incorrect since here it takes it as integer y

Output:

e
100

Objects

- **object:** An entity that contains data and behavior.
  - *data*: variables inside the object
  - *behavior*: methods inside the object
    - You interact with the methods; the data is hidden in the object.



- Constructing (creating) an object:

```
Type objectName = new Type(parameters);
```

- Calling an object's method:

```
objectName.methodName(parameters);
```

Methods

- **What is a Method (a function)?**

A *subprogram* (set of java statements) used to do a certain task.
A method has zero or more inputs (called parameters), and zero or one output (called return value)

- We will study ***Methods*** in more detail later.

- Example:

```
public static void main(String[] args)  {
        …
}
```

Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a **String** is not created with **new**.

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

  - Examples:

```
String name = "Marla Singer";
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

`String` objects

- A variable of type **`String`** is different from the other (primitive) data types we've seen so far
  - It is actually a reference to a **`String`** object .

- Examples:

```
String str = "hello there!";

int len = str.length();

String first = str.substring(0, 1);
```

Indexes

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

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

  - First character's index : 0
  - Last character's index : 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 not found)
`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* is 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 s = "Dr. Dre";
System.out.println(s.length());    // 7
```

`String` method examples

```
// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());
                              // 12
System.out.println(s1.indexOf("e"));
                             // 8
System.out.println(s1.substring(7, 10));
                             // "Reg"

String s3 = s2.substring(0, 7);
System.out.println(s3.toLowerCase());
                             // "marty s"
```

- Given the following string:

```
// index        01234567890123456789012
String book = "Building Java Programs";
```

  - How would you extract the word `"Java"` ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);
                  // lil bow wow
```

Strings are **immutable objects** which means that their values cannot be changed.

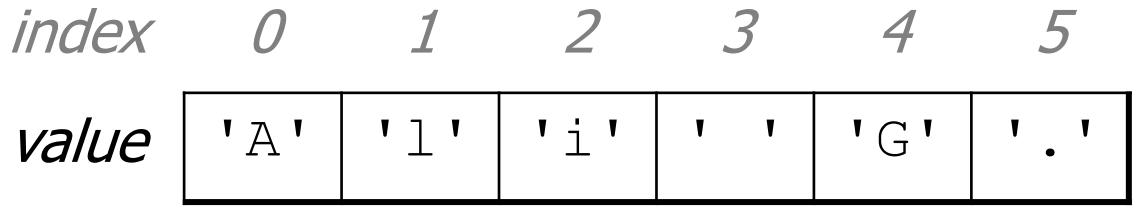
- To modify a variable's value, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);
                  // LIL BOW WOW
```

String and `char`

- A **`String`** is stored internally as an *array* of **`char`**

```
String s = "Ali G.";
```




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

The `charAt` method

- The `chars` in a `String` can be accessed using the ***`charAt`*** method.
  - accepts an **`int`** index parameter and returns the `char` at that index

```
String food = "cookie";
char firstLetter = food.charAt(0);   // 'c'

System.out.println(firstLetter + " is for " +
food);
```

Output:
c is for cookie

`char` vs. `String`

- **`"h"`** is a **`String`**, but **`'h'`** is a **`char`** (they are different)
- A `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'
```

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

```
char c = 'h';
c = c.toUpperCase();                  // ERROR
s = s.charAt(0).toUpperCase();        // ERROR
```

  - What is `s + 1`? What is `c + 1`?
  - What is `s + s`? What is `c + c`?

Math Library

Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`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.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`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

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

Calling `Math` methods

`Math.`**`methodName`**`(`**`parameters`**`)`

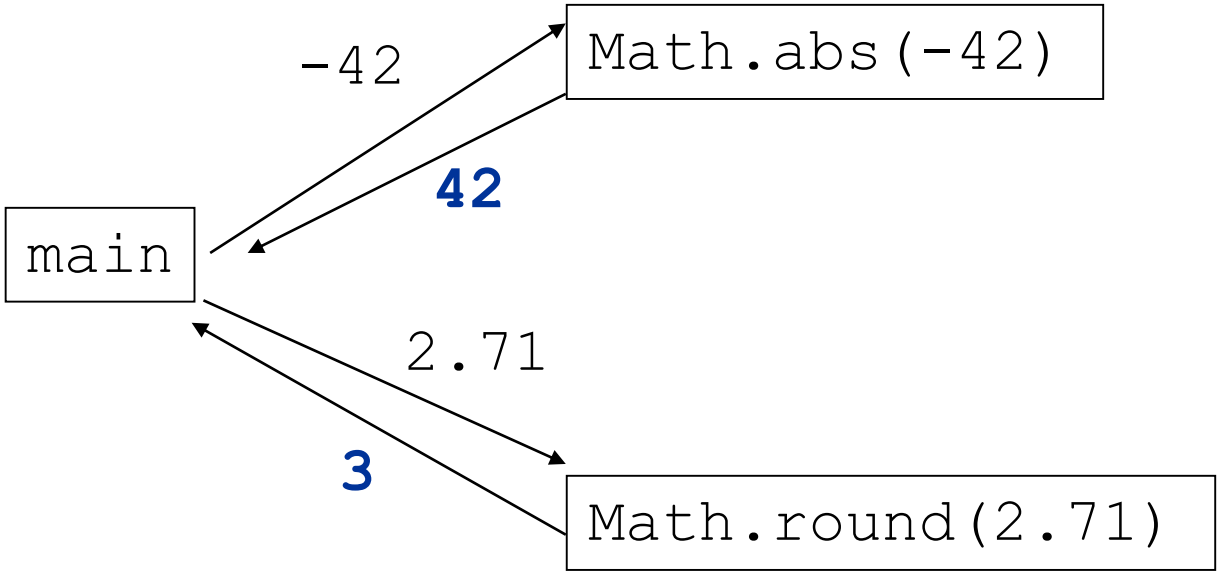
- Examples:

```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);              // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);           // 50

System.out.println(Math.min(3, 7) + 2);      // 5

double exp = Math.pow(2, 4);
System.out.println(exp);                     // 16.0
```

- The `Math` methods do not print to the console.
  - Each method produces ("returns") a numeric result.
  - The results are used as expressions (printed, stored, etc.).

Calling `Math` methods

- Parameters send information *in* from the caller to the method.
- Return values send information *out* from a method to its caller.
  - A call to the method can be used as part of an expression.



Remember: 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
```

`Math` questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

Generate a Random Variable

- Generate a random number between **low** and **high** (inclusive)

```
public static void main(String[] args)
    {
        int low, high, val;
        low = 10;
        high = 20;
        val = low + (int) (Math.random() * ((high - low) + 1));
        System.out.println(val); // 10, …, 20
}
```

Interactive Programs with *Scanner*

Input and `System.in`

- **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

- 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

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
`next().charAt(0)`	reads one `char` from the user
`nextLine()`	reads a one-*line* `String` from the user

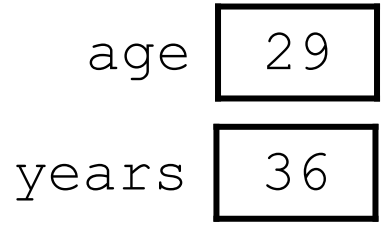
- Each method waits until the user presses Enter.
- The value typed by the user is returned.

```
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

```
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!");
    }
}
```



- Console (user input underlined):

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

Scanner example 2

```
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);
    }
}
```

- Output (user input underlined):

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

  - The Scanner can read multiple values from one line.

Scanner example 3

```
// Java program to find the Hypotenuse of a right angled triangle

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

public class Hypotenuse {
    public static void main(String[] args) {
         Scanner input = new Scanner(System.in);
         double side1, side2, hypotenuse;

        //get the length of the first side
         System.out.print("Please enter the value of Side1: ");
         side1 = input.nextDouble();
        //get the length of the second side
         System.out.print("Please enter the value of Side2: ");
         side2 = input.nextDouble();

       // calculate the hypotenuse
         hypotenuse = Math.sqrt(Math.pow(side1, 2) + Math.pow(side2,
  2));

          System.out.println("The length of the hypotenuse is: " +
  hypotenuse);
    }
}
```

- Output (user input underlined):

```
Please enter the value of Side1: 3
Please enter the value of Side2: 4
The length of the hypotenuse is 5
```

Input tokens

- **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)
        ...
```

Strings as user input

- `Scanner`'s `next` method reads a word of input as a `String`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your name? Chamillionaire
CHAMILLIONAIRE has 14 letters and starts with C
```

- The `nextLine` method reads a line of input as a `String`.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```