Welcome to SWE510!

Emre Ugur emre.ugur@boun.edu.tr @ BM33

https://www.cmpe.boun.edu.tr/~emre/courses/swe510/index.html

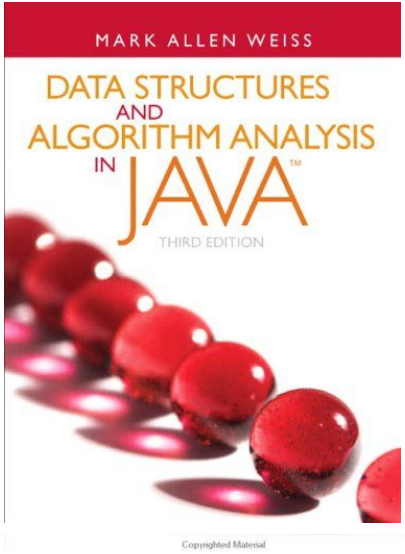
Communication: Through mailing list

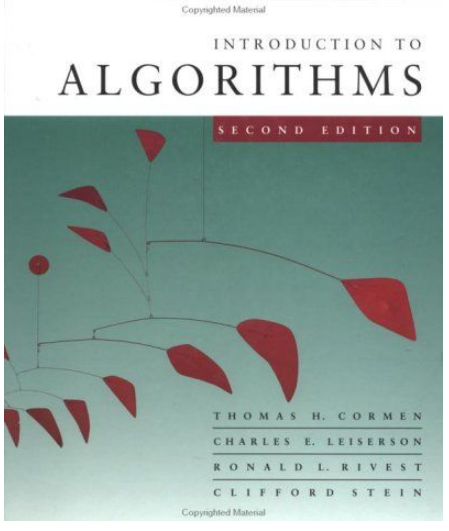
Grading:

- HW1: 5%
- HW2: 10%
- HW3: 10%
- Midterm 1: take-home exam, 15%
- Midterm 2: in-class exam, 20%
- Final, 30%
- In-class exercises: 10%

Bring your laptop

HW1: Install Eclipse to your laptop, and program...





Welcome to SWE510

Catalog Description:

Specification, usage and implementation and analysis of advanced data structures and algorithms. Hashing, heap structures, advanced sorting techniques, graphs and algorithm design techniques.

- Backgrounds, programming language, experience
  - With object-oriented languages
    - C++, Java, others?
  - With main data-structures and algorithms
    - List, queue, stack
    - Algorithm analysis (Big Oh notation)
    - Binary trees, non-binary trees
    - Sorting algorithms - quicksort, heapsort
    - Hashing
    - Graphs

Mar 2018	Mar 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.941%	-1.44%
2	2		C	12.760%	+5.02%
3	3		C++	6.452%	+1.27%
4	5	↑	Python	5.869%	+1.95%
5	4	↓	C#	5.067%	+0.66%
6	6		Visual Basic .NET	4.085%	+0.91%
7	7		PHP	4.010%	+1.00%
8	8		JavaScript	3.916%	+1.25%
9	12	↑	Ruby	2.744%	+0.49%
10	-	⇈	SQL	2.686%	+2.69%
11	11		Perl	2.233%	-0.03%
12	10	↓	Swift	2.143%	-0.13%
13	9	⇊	Delphi/Object Pascal	1.792%	-0.75%
14	16	↑	Objective-C	1.774%	-0.22%
15	15		Visual Basic	1.741%	-0.27%
16	13	↓	Assembly language	1.707%	-0.53%
17	17		Go	1.444%	-0.54%
18	18		MATLAB	1.408%	-0.45%
19	19		PL/SQL	1.327%	-0.34%
20	14	⇊	R	1.128%	-0.89%

https://adtmag.com/articles/2018/03/09/~/media/ECG/adtmag/Images/2018/03/tiobe_march_18.asxh

TIOBE

Frequency: Monthly.

Methodology: Based on the number of queries in popular search engines such as Google, Bing, Yahoo, Wikipedia, Amazon, YouTube, and Baidu using the +“<language> programming” search term.

Rankings:

1. Java
2. C
3. C++
4. C#
5. Python
6. JavaScript
7. PHP
8. Visual Basic .NET
9. Perl
10. Delphi
11. Ruby

Mar 2018	Mar 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.941%	-1.44%
2	2		C	12.760%	+5.02%
3	3		C++	6.452%	+1.27%
4	5	↑	Python	5.869%	+1.95%
5	4	↓	C#	5.067%	+0.66%
6	6		Visual Basic .NET	4.085%	+0.91%
7	7		PHP	4.010%	+1.00%
8	8		JavaScript	3.916%	+1.25%
9	12	↑	Ruby	2.744%	+0.49%
10	-	⇈	SQL	2.686%	+2.69%
11	11		Perl	2.233%	-0.03%
12	10	↓	Swift	2.143%	-0.13%
13	9	⇊	Delphi/Object Pascal	1.792%	-0.75%
14	16	↑	Objective-C	1.774%	-0.22%
15	15		Visual Basic	1.741%	-0.27%
16	13	↓	Assembly language	1.707%	-0.53%
17	17		Go	1.444%	-0.54%
18	18		MATLAB	1.408%	-0.45%
19	19		PL/SQL	1.327%	-0.34%
20	14	⇊	R	1.128%	-0.89%

https://adtmag.com/articles/2018/03/09/~/media/ECG/adtmag/Images/2018/03/tiobe_march_18.asxh

TIOBE

Frequency: Monthly.

Methodology: Based on the number of queries in popular search engines such as Google, Bing, Yahoo, Wikipedia, Amazon, YouTube, and Baidu using the +“<language> programming” search term.

Rankings:

1. Java
2. C
3. C++
4. C#
5. Python
6. JavaScript
7. PHP
8. Visual Basic .NET
9. Perl
10. Delphi
11. Ruby

Language Types (click to hide)

Web | Mobile | Enterprise | Embedded

The ranking is calculated using 12 weighted data sources. Click a data source to toggle its inclusion in the ranking and drag its slider to reweight it.

Use data from: **2018** | 2017 | 2016 | 2015 | 2014

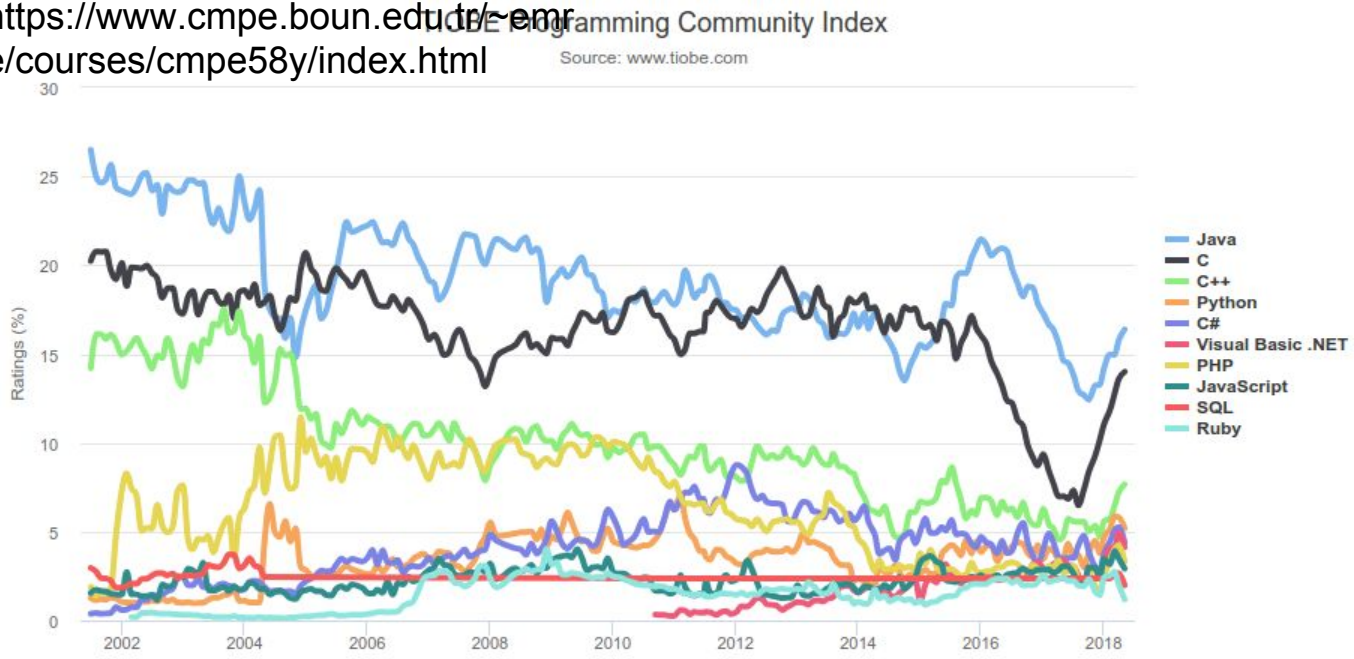
Google (search)	0	Google (trends)	0
Github (active)	100	**Github (created)**	100
Stack Overflow (?s)	0	Stack Overflow (views)	0
Reddit	0	Hacker News	0
Career Builder	100	Dice	0
Twitter	0	**IEEE Xplore**	100

Cancel | Save as Custom

Language Rank	Types	Custom Ranking
1. Java		100.0
2. Python		96.0
3. JavaScript		95.0
4. C#		94.5
5. C		94.5
6. C++		90.5
7. PHP		88.9
8. HTML		86.4
9. Matlab		79.4
10. Ruby		78.4

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

https://www.cmpe.boun.edu.tr/~emre/courses/cmpe58y/index.html



TIOBE programming community index is a measure of popularity of programming languages, created and maintained by the TIOBE Company based in Eindhoven, the Netherlands.

Java
Genel Bakış

Java nedir?

Java is a

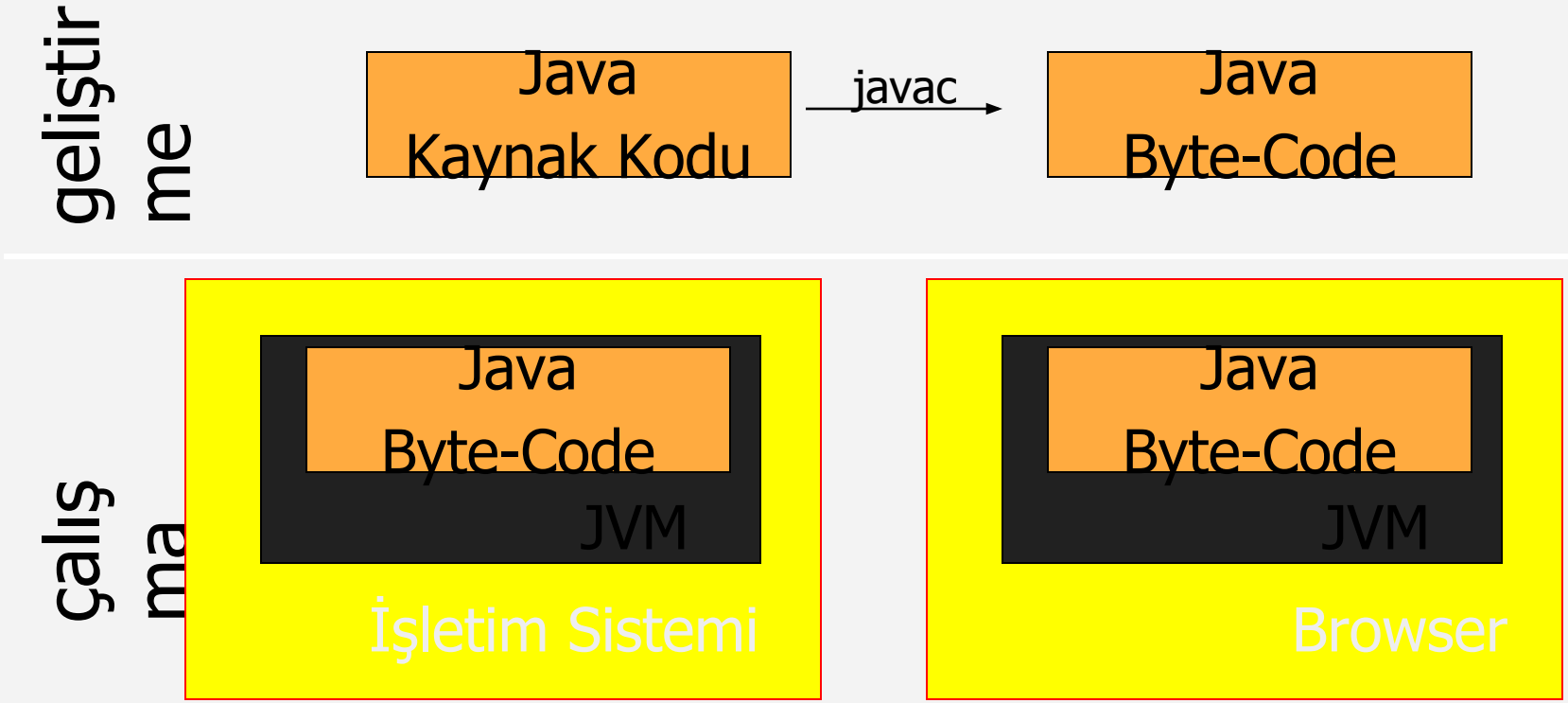
- simple,
- object oriented,
- distributed,
- interpreted,
- robust,
- secure,
- architecture natural,
- portable,
- high-performance,
- multithreaded,
- dynamic

language.

Java nedir? ...

- Sun geliştirdi (James Gosling)
  - 1991 Embedded sistemler
  - 1995 web
- Bir programlama dili
  - Nesneye dayalı
  - Concurrent
    - Thread
    - Synchronization
- Standart class kütüphaneleri

Java Çalışma Ortamı



Java Artılar

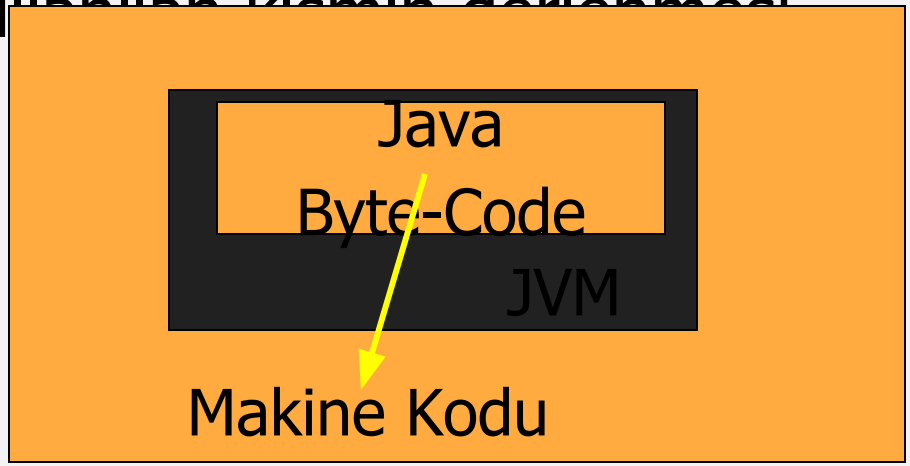
- Platformdan bağımsız
  - Windows, Unix, Mac
  - Internet Explorer, Netscape Navigator
- Standart kütüphaneler
  - Grafik Kullanıcı Arayüzü
  - Resim ve ses
  - JDBC
  - Network

Java Artılar ...

- Tasarım olarak internete hazır
  - "Canlı" Web sayfaları
  - FTP clients, POP mail clients
  - HTTP
  - RMI-Remote Method Invocation
  - Güvenlik
  - ...

JIT Derleyiciler

- Just In Time (JIT)
  - byte-code -> makine kodu
  - istemci tarafında derleme
  - sadece ku



Java ve C++ Karşılaştırması

- pointer
- struct, union, enum
- çoklu kalıtım
- preprocessor

Java ve C++ Karşılaştırması ...

- Byte-code
- String ve diziler birer nesne
- otomatik garbage collection

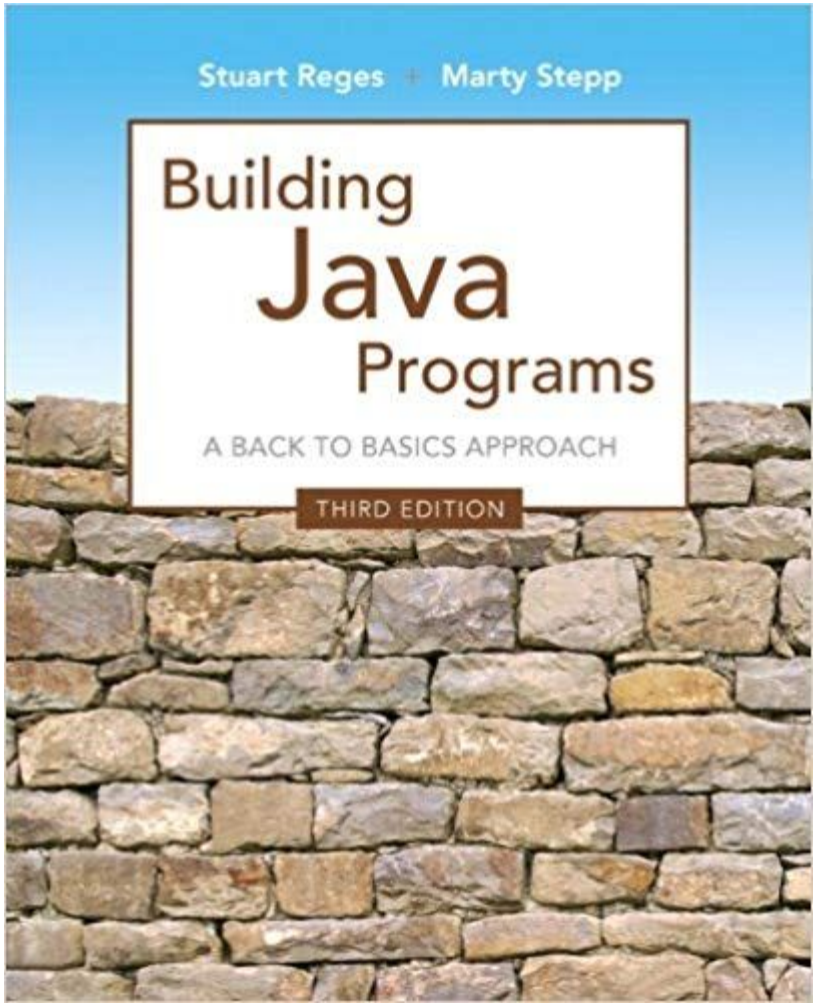
JDK Araçları

- Java Development Kit (JDK)
  - javac
    - Java derleyicisi
  - appletViewer
    - appletlerin çalıştığı ortam
  - java
    - uygulamaların çalıştığı ortam
  - javadoc
    - /** … */ belgelemesi

Entegre Geliştirme Ortamları

- eclipse (www.eclipse.org)
- JEdit (www.jedit.org)
- Visual Age for Java -> WebSphere
- Forte -> SunONE
- JBuilder
- ...

Introduction to Java



A Java program

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

```
Hello, world!

This program produces
four lines of output
```

Names and identifiers

- You must give your program a name.

```
public class GangstaRap {
```

  - **Naming convention: capitalize each word (e.g. MyClassName)**
  - **Your program's file must match exactly (GangstaRap.java)**
    - includes capitalization (Java is "case-sensitive")

- **identifier**: A name given to an item in your program.
  - **must start with a letter or _ or $**
  - **subsequent characters can be any of those or a number**
    - legal: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
    - illegal: `me+u` `49ers` `side-swipe` `Ph.D's`

Escape sequences

- **escape sequence:** A special sequence of characters used to represent certain special characters in a string.

`\t`	tab character
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

- **Example:**

```
System.out.println("\\hello\nhow\tare \"you\"?\\\\");
```

- **Output:**

```
\hello
how    are "you"?\\
```

Comments and Readability

- Free-format language
- One statement per line. İndent properly.
- Use blank lines.
- Comment: explain their code, compiler ignores
- Two comments: A slash and asterix
- /* Multiple lines ok */ /* not */ ok*/
- // short comments

PRIMITIVE DATA TYPE in JAVA

Type	Contains	Default	Size	Range
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
float	IEEE 754 floating point	0.0f	32 bits	±1.4E-45 to ±3.4028235E+38
long	Signed integer	0L	64 bits	-9223372036854775808 to 9223372036854775807
double	IEEE 754 floating point	0.0d	64 bits	±4.9E-324 to ±1.7976931348623157E+308
boolean	true or false	FALSE	1 bit	NA
char	Unicode character	'\u0000'	16 bits	\u0000 to \uFFFF

Arithmetic operators

- **operator: Combines multiple values or expressions.**
  - **`+` addition**
  - **`-` subtraction (or negation)**
  - **`*` multiplication**
  - **`/` division**
  - **`%` modulus (a.k.a. remainder)**

- **As a program runs, its expressions are *evaluated*.**
  - **`1 + 1` evaluates to `2`**
  - **`System.out.println(3 * 4);` prints `12`**
    - How would we print the text `3 * 4` ?

Precedence

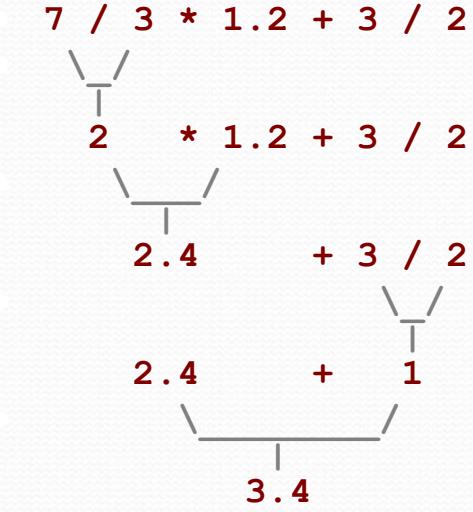
- **precedence: Order in which operators are evaluated.**
  - **Generally operators evaluate left-to-right.**
    1 - 2 - 3 is (1 - 2) - 3 which is -4
  - **But */% have a higher level of precedence than +-**

```
  1 + 3 * 4  is 13
6 + 8 / 2 * 3
6 +   4   * 3
6 +     12   is 18
```

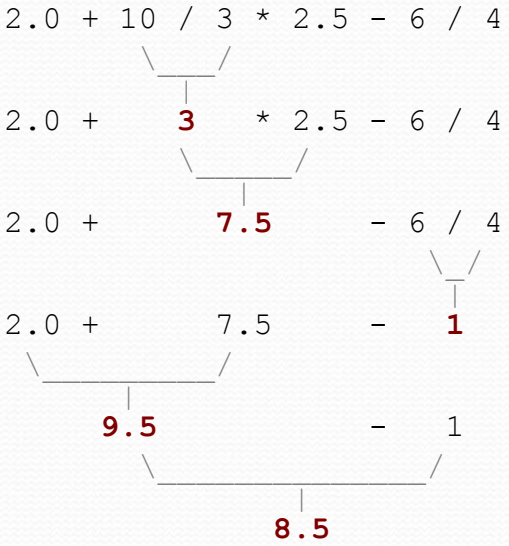
  - **Parentheses can force a certain order of evaluation:**
    (1 + 3) * 4is 16
  - **Spacing does not affect order of evaluation**
    1+3 * 4-2 is 11

Mixing types

- When int and double are mixed, the result is a double. 4.2 * 3 is 12.6
- The conversion is per-operator, affecting only its operands.



3 / 2 is 1 above, not 1.5.



For loop

```
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

Nested loops

- **nested loop**: A loop placed inside another loop.

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.print((i * j) + "\t");
    }
    System.out.println();   // to end the line
}
```

- Output:

```
1    2    3    4    5
2    4    6    8    10
3    6    9    12   15
4    8    12   16   20
```

- Statements in the outer loop's body are executed 4 times.
  - The inner loop prints 5 numbers each time it is run.

Pseudo-code

- **pseudo-code**: An English description of an algorithm.

- Example: Drawing a 12 wide by 7 tall box of stars

```
print 12 stars.
for (each of 5 lines) {
    print a star.
    print 10 spaces.
    print a star.
}
print 12 stars.
```

```
************
*          *
*          *
*          *
*          *
*          *
************
```

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
```

scope of i

scope of x

Scope implications

- Variables without overlapping scope can have same name.

```
for (int i = 1; i <= 100; i++) {
    System.out.print("/");
}
for (int i = 1; i <= 100; i++) {    // OK
    System.out.print("\\");
}
int i = 5;                          // OK: outside of loop's scope
```

- A variable can't be declared twice or used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                      // ERROR: overlapping scope
    System.out.print("/");
}
i = 4;                              // ERROR: outside 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;
```

Declaring a parameter

Stating that a method requires a parameter in order to run

```
public static void name ( <Type> <name> ) {
    statement(s);
}
```

- Example:

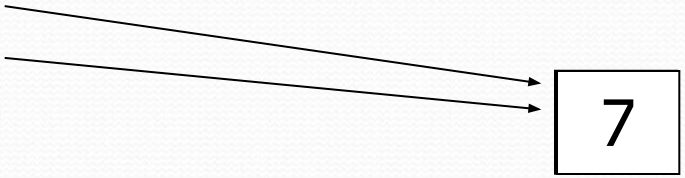
```
public static void sayPassword(int code) {
    System.out.println("The password is: " + code);
}
```

  - *When sayPassword is called, the caller must specify the integer code to print.*

How parameters are passed

- When the method is called:
  - *The value is stored into the parameter variable.*
  - *The method's code executes using that value.*

```
public static void main(String[] args) {
    chant(3);
    chant(7);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```



Multiple parameters example

```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}

Output:
```

Value semantics

- value semantics: When primitive variables (int, double) are passed as parameters, their values are copied.
  - *Modifying the parameter will not affect the variable passed in.*

```
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}

public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);
    ...
}
```

Strings

- **string: A sequence of text characters.**

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

  - ***Examples:***

```
String name = "Marla Singer";

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

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...

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
```

- ***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.).

Return examples

```
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}

//You can shorten the examples by returning an expression:
    public static double fToC(double degreesF) {
        return 5.0 / 9.0 * (degreesF - 32);
    }
```

More about type casting

- *Type casting has high precedence and only casts the item immediately next to it.*
  - `double x = (double) 1 + 1 / 2;  // 1`
  - `double y = 1 + (double) 1 / 2;  // 1.5`
- *You can use parentheses to force evaluation order.*
  - `double average = (double) (a + b + c) / 3;`
- *A conversion to double can be achieved in other ways.*
  - `double average = 1.0 * (a + b + c) / 3;`

System.out.printf

an advanced command for printing formatted text

System.out.printf("format string", parameters);

- *A format string contains placeholders to insert parameters into it:*
  - **%dan integer**
  - **%fa real number**
  - **%sa string**

  - **Example:**

```
int x = 3;
int y = 2;
System.out.printf("(%d, %d)\n", x, y);   // (3, 2)
```

Class - Object

Object

A programming entity that contains state (data) and behavior (methods).

Class

A category or type of object.

Input and `System.in`

- `System.out`
  - An object with methods named println and print
- `System.in`
  - not intended to be used directly
  - We use a second object, from a class Scanner, to help us.
- Constructing a Scanner object to read console input:

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

Java class libraries, import

- Java class libraries: Classes included with Java's JDK.
  - organized into groups named *packages*
  - To use a package, put an *import declaration* in your program.
- Syntax:

```
// put this at the very top of your program
import packageName.*;
```

- Scanner is in a package named java.util

```
import java.util.*;
```

  - To use Scanner, you must place the above line at the top of your program (before the public class header).

Scanner methods

Method	Description
`nextInt()`	reads a token of user input as an `int`
`nextDouble()`	reads a token of user input as a `double`
`next()`	reads a token of user input as a `String`
`nextLine()`	reads a *line* of user input as a `String`

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

```
System.out.print("How old are you? ");     // prompt
int age = console.nextInt();
System.out.println("You'll be 40 in " +
        (40 - age) + " years.");
```

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

Example `Scanner` usage

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

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

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

        System.out.println(age + "... That's quite old!");
    }
}
```

- Output (user input underlined):

```
How old are you? 14
14... That's quite old!
```

Example `Scanner` usage

```
import java.util.*;    // so that I can use Scanner
public class ReadUntilLargerThan40 {
        String prompt="Enter a number, program terminates if it is larger than 40: ";
    Scanner s = new Scanner(System.in);
    System.out.println(prompt);
        for (int x=s.nextInt();x<=40;x=s.nextInt() ){
                System.out.println(prompt);
        }
        System.out.println("Finished!");
}
```

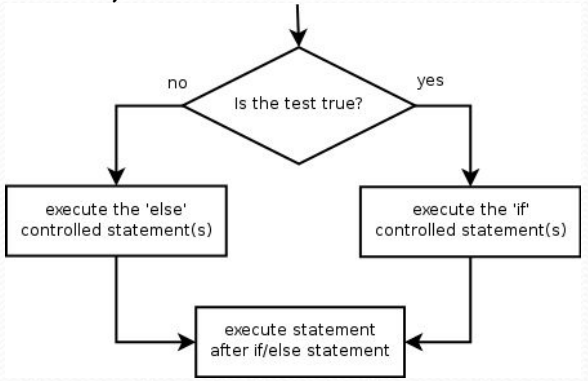
- **Output (user input underlined)**

  Enter a number, program terminates if it is larger than 40:
  -20
  Enter a number, program terminates if it is larger than 40:
  55
  Finished!

The `if/else` statement

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.");
}
```

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

Evaluating logic expressions

- Relational operators have lower precedence than math.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35  >= 3 + 30
35  >= 33
true
```

- Relational operators cannot be "chained" as in algebra.

```
    2 <= x <= 10              (assume that x is 15)
    true   <= 10
       error!
```

Instead, combine multiple tests with && or ||

```
    2 <= x && x <= 10         (assume that x is 15)
```

All paths must return

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    }   // Error: not all paths return a value
}
```

- The following also does not compile:

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= a && b >= c) {
        return b;
    } else if (c >= a && c >= b) {
        return c;
    }
```

Objects and classes

- **object: An entity that contains:**
  - ***data* (variables), and**
  - ***behavior* (methods).**

- **class: A program, or a type of objects.**

- **Examples:**
  - **The class String represents objects that store text.**
  - **The class DrawingPanel represents graphical window objects.**
  - **The class Scanner represents objects that read information from the keyboard, files, and other sources.**

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;
```

Indexes

- 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	y

- 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 (seen

`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` 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(2, 8);
    System.out.println(s3.toLowerCase());     // "rty st"
```

Given the following string:

```
    //       index 0123456789012345678901
    String book = "Building Java Programs";
```

- **How would you extract the word "Java" ?**
- **How would you extract the first word from any string?**

Modifying strings

- **Methods like substring, toLowerCase, etc. create/return a new string, rather than modifying the current string.**

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

- **To modify a variable, you must reassign it:**

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

Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");

        String teacher = "Helene";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

```
Output:
Welcome, Marty
Welcome, Helene
```

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? Madonna
MADONNA has 7 letters and starts with M
```

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

```
System.out.print("What is your address? ");
```

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.**

`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 `charAt` method

- **The chars in a String can be accessed using the charAt method.**

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

System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- **You can use a for loop to print or examine each character.**

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
      char c = major.charAt(i);
      System.out.println(c);
}
```

Output:

C
S
E

char **vs.** int

- **All char values are assigned numbers internally by the computer, called *ASCII* values.**
  - **Examples:**

```
'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'
```

Comparing `char` values

- **You can compare char values with relational operators:**

```
'a' < 'b'    and 'X' == 'X'    and 'Q' != 'q'
```

  - **An example that prints the alphabet:**

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

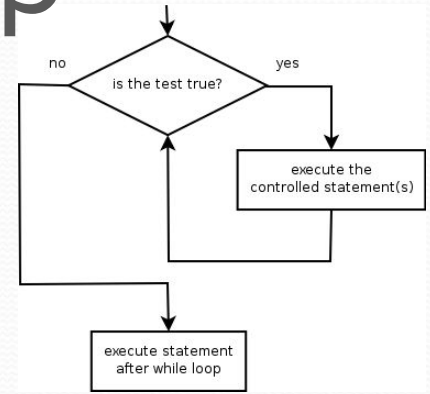
- **You can test the value of a string's character:**

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
}
```

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while (test) {
    statement(s);
}
```



- Example:

```
int num = 1;                    // initialization
    while (num <= 200) {            // test
        System.out.print(num + " ");
        num = num * 2;              // update
    }
```

  - OUTPUT:

```
1 2 4 8 16 32 64 128
```

break

- **`break`** statement: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - Such loops are often called *"forever" loops* because their header's boolean test is often changed to a trivial `true`.

```
while (true) {
    statement(s);

    if (test) {
        break;
    }

    statement(s);
}
```

  - `break` is bad style!

continue

- The continue statement immediately ends the current iteration of a loop.
- The code execution will return to the loop's header, which will perform the loop's test again (along with the update step, if it is a for loop).
- The following loop uses continue to avoid negative integers:

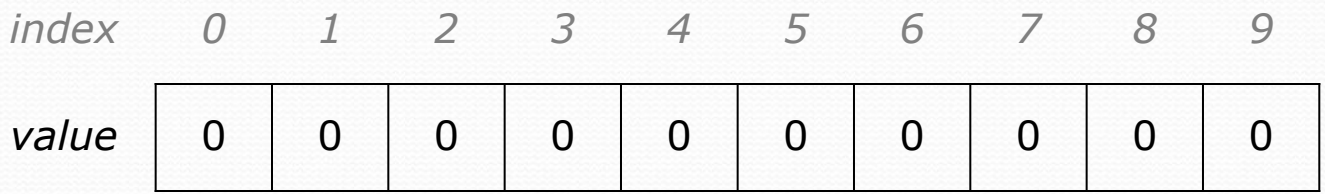
```
int sum = 0;
for (int i = 0; i < 100; i++) {
     System.out.print("type a nonnegative integer:");
     int number = console.nextInt();
     if (number < 0) {
          continue; // skip this number
     }
     sum += number;
}
```

Array declaration

type`[]` **name** `=` `new` **type**`[`**length**`];`

- Example:

```
int[] numbers = new int[10];
```



Array declaration, cont.

- The length can be any integer expression.

```
int x = 2 * 3 + 1;
int[] data = new int[x % 5 + 2];
```

- Each element initially gets a "zero-equivalent" value.

Type	Default value
`int`	`0`
`double`	`0.0`
`boolean`	`false`
`String` or other object	`null` (means, "no object")

Accessing elements

```
name[index]   // access
name[index] = value;    // modify
```

- Example:

```
int[] numbers = new int[10];
    numbers[0] = 27;
    numbers[3] = -6;
    System.out.println(numbers[0]);
    if (numbers[3] < 0) {
        System.out.println("Element 3 is negative.");
    }
```

index	0	1	2	3	4	5	6	7	8	9
value	**27**	0	0	**-6**	0	0	0	0	0	0

The `length` field

- An array's `length` field stores its number of elements.

  name`.length`

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// output: 0 2 4 6 8 10 12 14
```

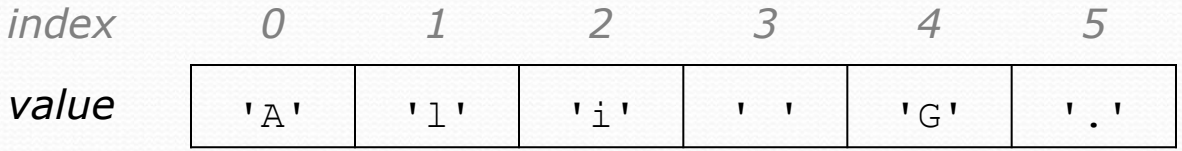
  - It does not use parentheses like a String's `.length()`.
- What expressions refer to:
  - The last element of any array?
  - The middle element?

Text processing

- **text processing**: Examining, editing, formatting text.
  - Often involves `for` loops to examine each letter of a `String`.
    - Count the number of times the letter 's' occurs in a file.
    - Find which letter is most common in a file.
    - Count A, C, T and Gs in `String`s representing DNA strands.

- `String`s are represented internally as arrays of `char`.

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



Recall: type `char`

- **`char`**: A primitive type representing a single character.
  - Values are surrounded with apostrophes: `'a'` or `'4'` or `'\n'`
- Access a string's characters with its `charAt` method.

```
String word = console.next();
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("That's good enough for me!");
}
```

- Use `for` loops to examine each character.

```
String coolMajor = "CSE";
for (int i = 0; i < coolMajor.length(); i++) {
    System.out.println(coolMajor.charAt(i));
}
```

Arrays.toString

- `Arrays.toString` accepts an array as a parameter and returns a `String` representation of its elements.

```
int[] e = {0, 2, 4, 6, 8};
e[1] = e[3] + e[4];
System.out.println("e is "+Arrays.toString(e));
```

Output:

```
e is [0, 14, 4, 6, 8]
```

- Must `import java.util.*;`

The `Arrays` class

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a sorted array (< 0 if not found)
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`**array**`, `**value**`)`	sets every element in the array to have the given value
`sort(`**array**`)`	arranges the elements in the array into ascending order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, 17]"`

"Array mystery" problem

- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

"Array mystery" problem

- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++) {
    if (a[i] > a[i + 1]) {
        a[i + 1] = a[i + 1] * 2;
    }
}

for (int i = 0; i < a.length - 1; i++)

   System.out.println(a[i]);
```

https://tinyurl.com/y7aa6w7j

The `Arrays` class - copyOf

```
int[] data = new int[2];
data[0] = 0; data[1] = 1;
int[] newData = data;
data[0] = 1;
System.out.println(newData[0]);
int[] newData = Arrays.copyOf(data, 2 * data.length);
```

Description

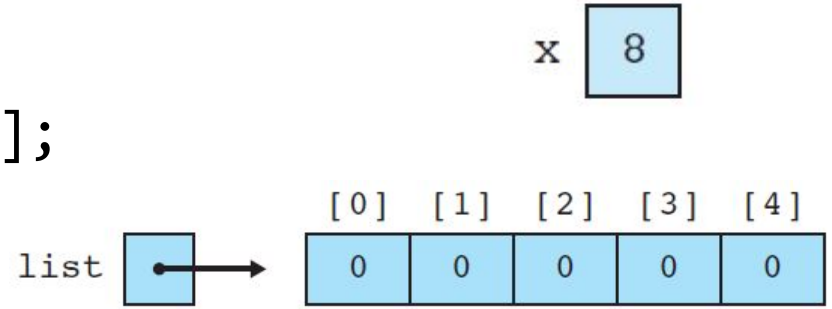
The **java.util.Arrays.copyOf(boolean[] original,int newLength)** method copies the specified array, truncating or padding with false (if necessary) so the copy has the specified length. For all indices that are valid in both the original array and the copy, the two arrays will contain identical values. For any indices that are valid in the copy but not the original, the copy will contain false. Such indices will exist if and only if the specified length is greater than that of the original array.

Arrays.equals

```
int[] data1 = {1, 1, 2, 3, 5, 8, 13, 21};
int[] data2 = {1, 1, 2, 3, 5, 8, 13, 21};

if (data1==data2) {
System.out.println("They store the same data");
}
if (Arrays.equals(data1, data2)) {
System.out.println("They store the same data");
}
```

Reference Semantics

```
int x = 8;
int[] list = new int[5];
```



We say that list refers to the array.

Reference Semantics

Value Semantics (Value Types)

A system in which values are stored directly and copying is achieved by creating independent copies of values. Types that use value semantics are called value types.

Reference Semantics (Reference Types)

A system in which references to values are stored and copying is achieved by copying these references. Types that use reference semantics are called reference types.

Reference Semantics

```
int[] list1 = new int[5];
int[] list2 = new int[5];
for (int i = 0; i < list1.length; i++) {
  list1[i] = 2 * i + 1;
  list2[i] = 2 * i + 1;
}
int[] list3 = list2;
```

Reference Semantics

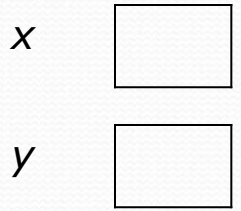
`null`

A Java keyword signifying no object.

```
String s = null;
int[] list = null;
```

Value semantics (primitives)

- **value semantics**: Behavior where values are copied when assigned to each other or passed as parameters.
  - When one primitive variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

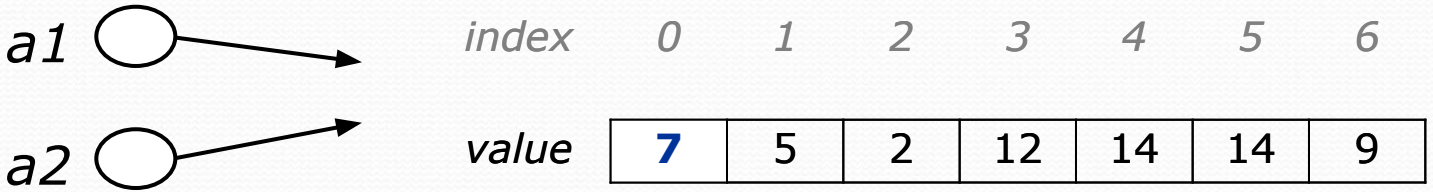



```
int x = 5;
int y = x;      // x = 5, y = 5
y = 17;         // x = 5, y = 17
x = 8;          // x = 8, y = 17
```

Reference semantics (objects)

- **reference semantics**: Behavior where variables actually store the address of an object in memory.
  - When one reference variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.
  - Modifying the value of one variable *will* affect others.

```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1;      // refer to same array as a1
a2[0] = 7;
System.out.println(a1[0]);   // 7
```



Reference Semantics

```
int[] list1 = new int[5];
int[] list2 = new int[5];
for (int i = 0; i < list1.length; i++) {
  list1[i] = 2 * i + 1;
  list2[i] = 2 * i + 1;
}
int[] list3 = list2;

for (int i=0;i<5;i++)
  System.out.print(list1[i]);
for (int i=0;i<5;i++)
  System.out.print(list2[i]);
for (int i=0;i<5;i++)
  System.out.print(list3[i]);
```

Reference Semantics

```
Exception: String!

and others....
```

Reference Semantics

```
int[] list1 = new int[5];
int[] list2 = new int[5];
for (int i = 0; i < list1.length; i++) {
  list1[i] = 2 * i + 1;
  list2[i] = 2 * i + 1;
}
int[] list3 = list2;

for (int i=0;i<5;i++)
  System.out.print(list1[i]);
for (int i=0;i<5;i++)
  System.out.print(list2[i]);
for (int i=0;i<5;i++)
  System.out.print(list3[i]);
```

Arrays as parameters

- Declaration:

```
public static type methodName(type[] name) {
```

  - Example:

```
public static double average(int[] numbers) {
```

- Call:

```
methodName(arrayName);
```

  - Example:

```
int[] scores = {13, 17, 12, 15, 11};
double avg = average(scores);
```

Array parameter example

```
public static void main(String[] args) {
    int[] iq = {126, 84, 149, 167, 95};
    double avg = average(iq);
    System.out.println("Average = " + avg);
}
public static double average(int[] array) {
    int sum = 0;
    for (int i = 0; i < array.length; i++) {
        sum += array[i];
    }
    return (double) sum / array.length;
}
```

Output:

```
Average = 124.2
```

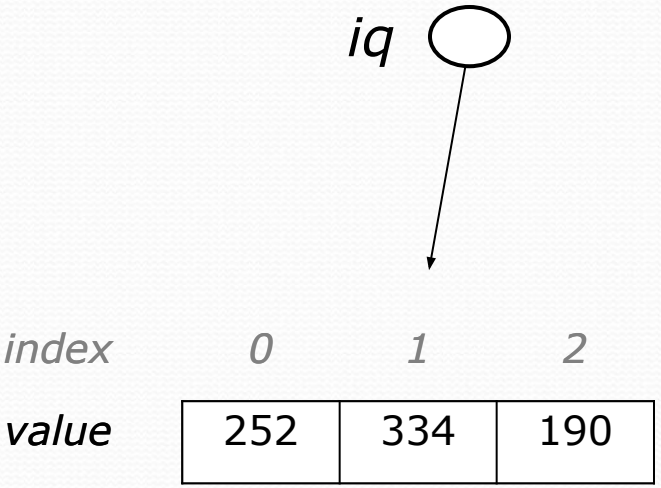
Reference Semantics

```
public static void incrementAll(int[] data) {
    for (int i = 0; i < data.length; i++)
        data[i]++;

    System.out.println(data[0]);
}

public static void incrementVal(int data) {
    data++;

    System.out.println(data);
}
```

Arrays passed by reference

- Arrays are objects.
  - When passed as parameters, they are passed by *reference.* (Changes made in the method are also seen by the caller.)
- Example:

```
public static void main(String[] args) {
    int[] iq = {126, 167, 95};
    doubleAll(iq);
    System.out.println(iq[0]);
}

public static void doubleAll(int[] a) {
    for (int i = 0; i < a.length; i++) {
        a[i] = a[i] * 2;
```



Arrays as return (declaring)

```
public static type[] methodName(parameters) {
```

Arrays as return

A method to compute both mod and div

```
public static int[] modDiv(int a, int b){
 int mod = a%b;
 int div = a/b;
 int[] retVal={mod, div};
 System.out.println(retVal);
 return retVal;
}
public static void main(String[] strList){
 int[] x = modDiv(21,4);
 System.out.println(x[0]+" "+x[1]);
 System.out.println(x);
```

Arrays as return (declaring)

```
public static type[] methodName(parameters) {
```

- Example:

```
public static int[] countDigits(int n) {
        int[] counts = new int[10];
        while (n > 0) {
            int digit = n % 10;
            n = n / 10;
            counts[digit]++;
        }
        return counts;
  }
```

Arrays as return (calling)

```
type[] name = methodName(parameters);
```

- Example:

```
public static void main(String[] args) {
    int[] tally = countDigits(229231007);
    System.out.println(Arrays.toString(tally));
}
```

Output:

```
[2, 1, 3, 1, 0, 0, 0, 1, 0, 1]
```

Garbage Collector

A process that is part of the Java Runtime Environment that periodically frees the memory used by objects that are no longer referenced.

```
public static int[] modDiv(int a, int b){
 int mod = a%b;
 int div = a/b;
 int[] retVal={mod, div};
 System.out.println(retVal);
 return retVal;
}
public static void main(String[] strList){
 int[] x = modDiv(21,4);
 System.out.println(x[0]+" "+x[1]);
 System.out.println(x);
```

Multi-dimensional arrays

Arrays of more than one dimension are called *multidimensional arrays*.

Multidimensional Array

An array of arrays, the elements of which are accessed with multiple integer indexes.

`double` : one double

`double[]` : a one-dimensional array of doubles

`double[][]` : a two-dimensional grid of doubles

`double[][][]`: a three-dimensional collection of doubles

```
double[][] temps = new double[3][5];
```

Multi-dimensional arrays

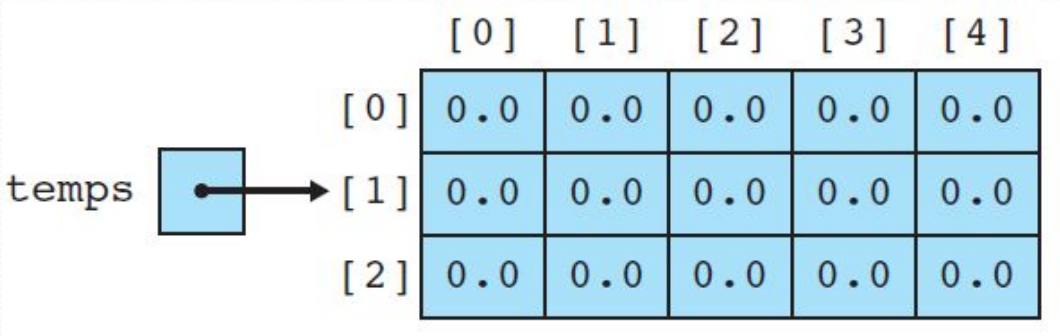
`double` : one double

`double[]` : a one-dimensional array of doubles

`double[][]` : a two-dimensional grid of doubles

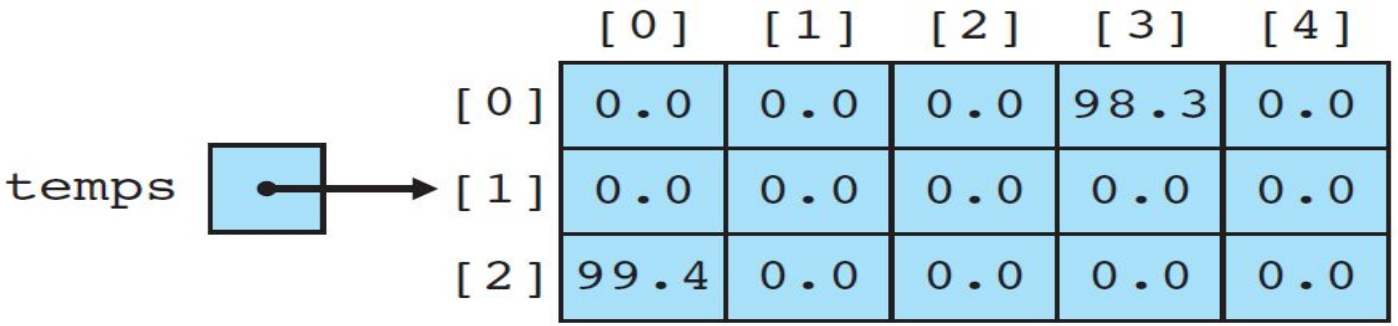
`double[][][]`: a three-dimensional collection of doubles

```
double[][] temps = new double[3][5];
```



Multi-dimensional arrays

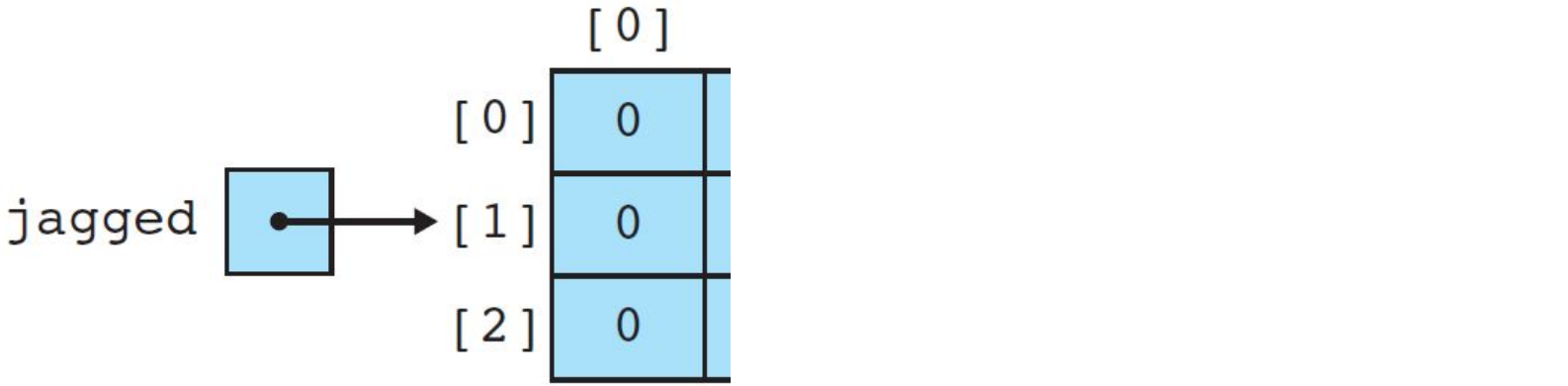
```
double[][] temps = new double[3][5];
temps[0][3] = 98.3; // fourth value of 1st row
temps[2][0] = 99.4; // first value of 3rd row
```



`temps`	the entire grid
`temps[2]`	the entire third row
`temps[2][0]`	the first element of the third row

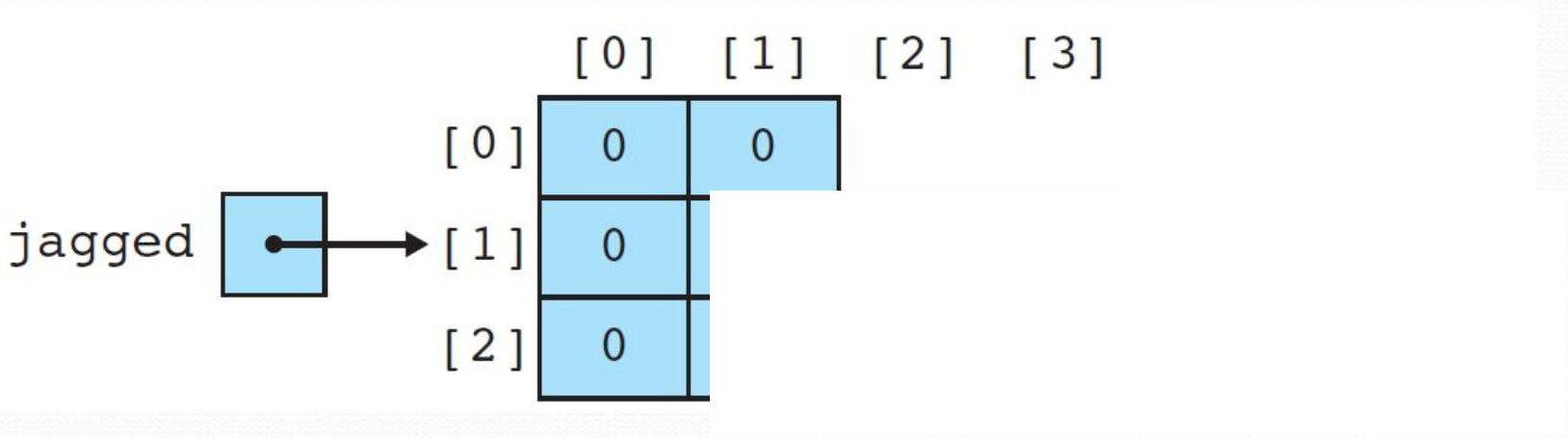
Jagged arrays

```
int [][] jagged = new int[3][];
```



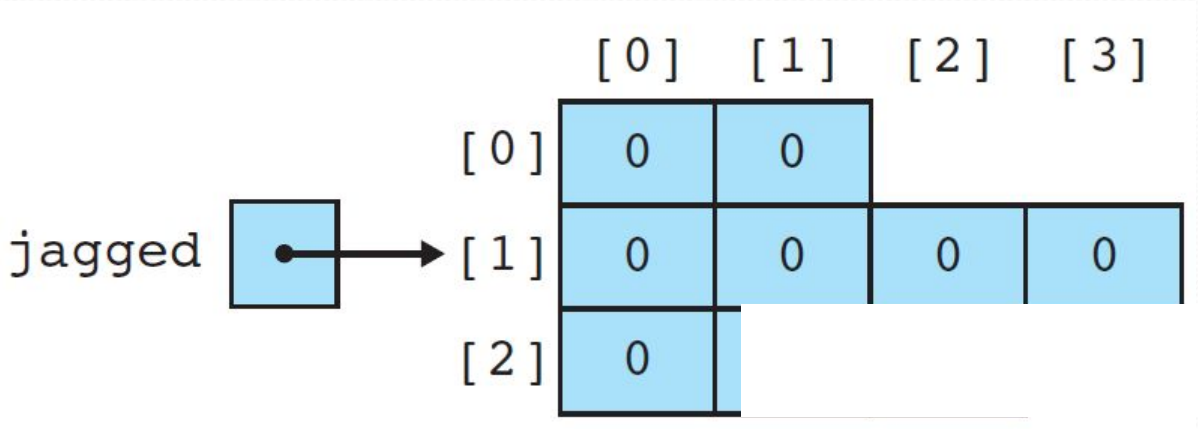
Jagged arrays

```
int [][] jagged = new int[3][];
jagged[0] = new int[2];
```



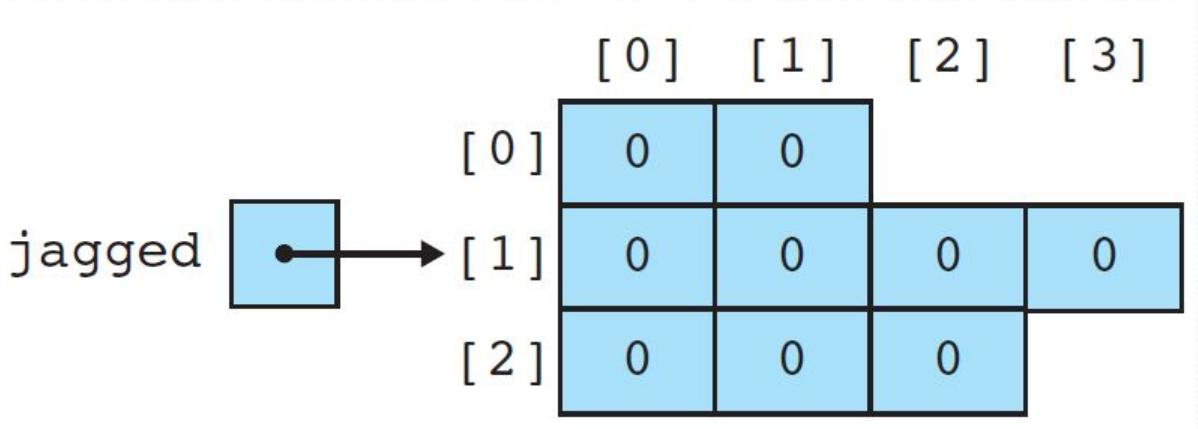
Jagged arrays

```
int [][] jagged = new int[3][];
jagged[0] = new int[2];
jagged[1] = new int[4];
```



Jagged arrays

```
int [][] jagged = new int[3][];
jagged[0] = new int[2];
jagged[1] = new int[4];
jagged[2] = new int[3];
```



Recursion

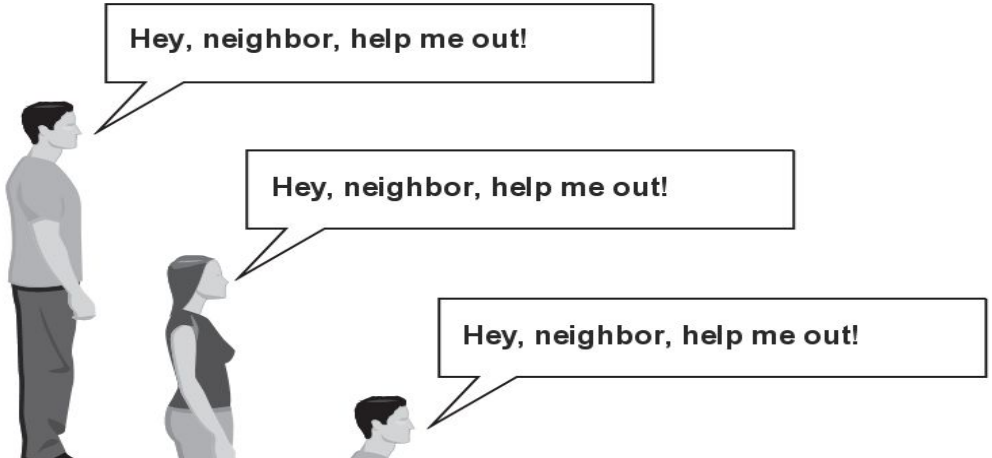
- **recursion**: The definition of an operation in terms of itself.
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.

- **recursive programming**: Writing methods that call themselves to solve problems recursively.

  - An equally powerful substitute for *iteration* (loops)
  - Particularly well-suited to solving certain types of problems

The idea

- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
  - Each person can solve a small part of the problem.
    - What is a small version of the problem that would be easy to answer?
    - What information from a neighbor might help me?



pow() method

pow(3, 4) returns 81

implement this method without using for loops

```
public static double pow(double base, double exponent){
    if (exponent==0)
        return 1;
    if (exponent<0)
        return (1/base)*pow(base,exponent+1);
    else
        return base*pow(base,exponent-1);
   }
```

isPalindrome

Write a recursive method *isPalindrome* accepts a String and returns true if it reads the same forwards as backwards.

- isPalindrome("madam") → true
- isPalindrome("racecar") → true
- isPalindrome("step on no pets")→ true
- isPalindrome("able was I ere I saw elba") → true
- isPalindrome("Java") → false
- isPalindrome("rotater") → false
- isPalindrome("byebye") → false
- isPalindrome("notion")→ false

isPalindrome

```
public static boolean isPalindrome(String str){
    if (str.length()==0 || str.length()==1)
        return true;
    if (str.charAt(0)!=str.charAt(str.length()-1))
        return false;
    return (isPalindrome(str.substring(1,str.length()-1)));
   }
```

```
public static boolean isPalindrome(String str){
    return ((str.length()==0 || str.length()==1) ||
           ((str.charAt(0)==str.charAt(str.length()-1)) &&
            (isPalindrome(str.substring(1,str.length()-1)))));
```