

Introduction to Computers, Programs, and Java

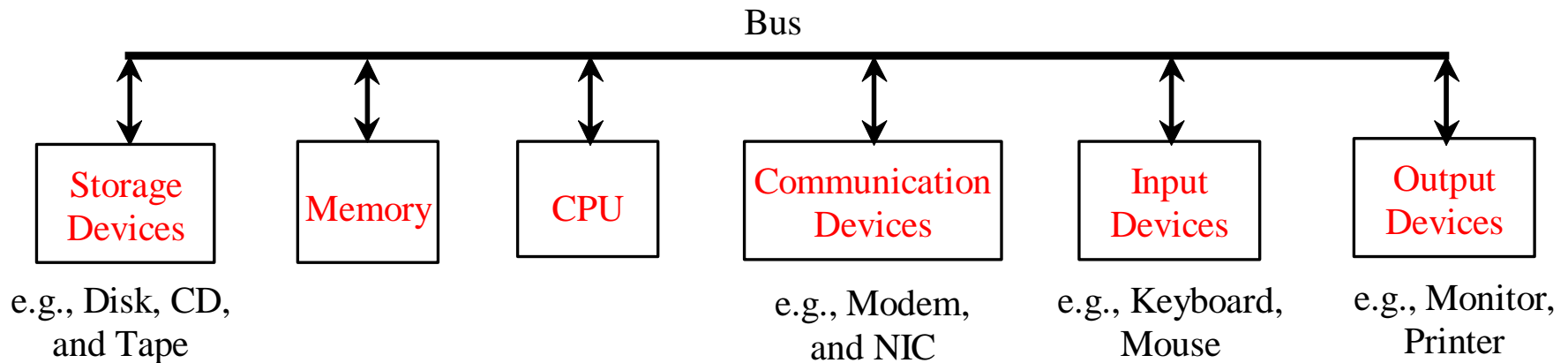
CSE 114, Computer Science 1

Stony Brook University

<http://www.cs.stonybrook.edu/~cse114>

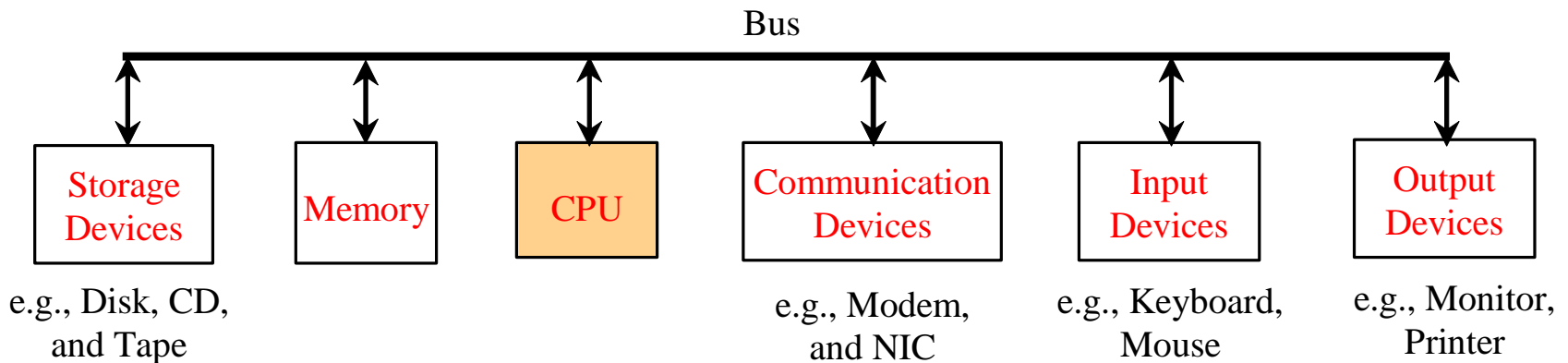
What is a Computer?

A computer consists of a CPU, memory, hard disk, monitor, printer, and communication devices.



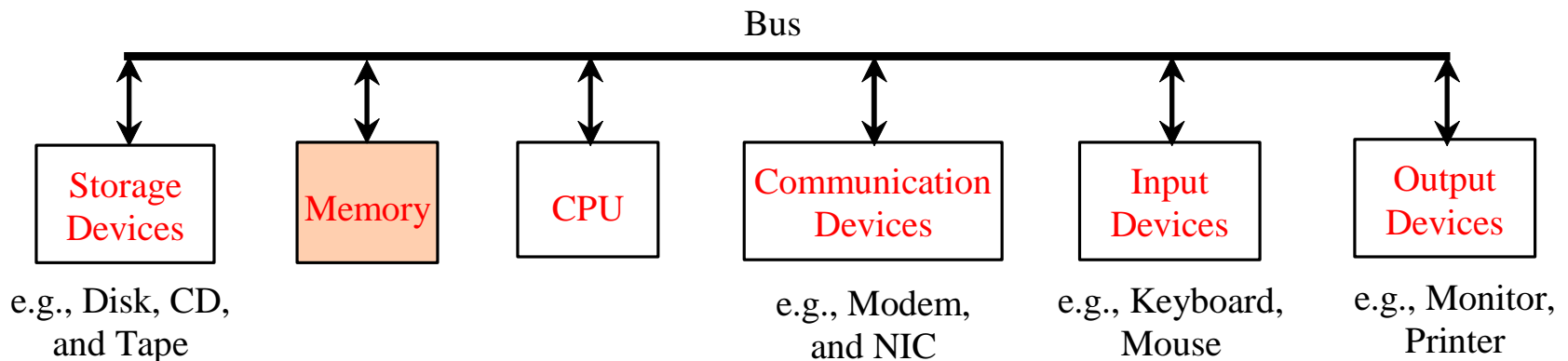
CPU

- Central Processing Unit (CPU)
 - retrieves instructions from memory and executes them
 - the CPU speed is measured in hertz = cycles per second (Hz, MHz = MegaHertz, GHz = Gigahertz)
 - 1 megahertz = 1 million pulses per second



Memory

- Stores data and program instructions for CPU to execute
 - ordered sequence of bytes (8 bits – binary base unit)



How Data is Stored?

- What's binary?
 - a base-2 number system
- What do humans use?
 - base-10
 - Why?
- Why do computers like binary?
 - electronics
 - easier to make hardware that stores and processes binary numbers than decimal numbers
 - more efficient: space & cost

Memory address	Memory content	
.	.	
.	.	
.	.	
2000	01001010	Encoding for character 'J'
2001	01100001	Encoding for character 'a'
2002	01110110	Encoding for character 'v'
2003	01100001	Encoding for character 'a'
2004	00000011	Encoding for number 3

Number Systems

- The digits in the **decimal number system** are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
 - A decimal number is represented using a sequence of one or more of these digits.
 - The value that each digit in the sequence represents depends on its position.
 - A position in a sequence has a value that is an integral power of 10.
 - e.g., the digits 7, 4, 2, and 3 in decimal number **7423** represent **7000**, **400**, **20**, and **3**, respectively:
- | | | | |
|--------|--------|--------|--------|
| 7 | 4 | 2 | 3 |
| 10^3 | 10^2 | 10^1 | 10^0 |
- $$= 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$
- $$= 7000 + 400 + 20 + 3 = 7423$$
- We say that 10 is the *base* or *radix* of the decimal number system.
 - The base of the binary number system is 2 since the binary number system has two digits
 - The base of the hex number system is 16 since the hex number system has sixteen digits.

Number Systems

- Computers use binary numbers internally because storage devices like memory and disk are made to store **0s and 1s**.
 - A number or a text inside a computer is stored as a sequence of 0s and 1s.
 - Each 0 and 1 is called a **bit** (short for **binary digit**)
- Binary numbers are not intuitive, since we use decimal numbers in our daily life.
 - When you write a number like 20 in a program, it is assumed to be a decimal number.
 - Internally, computer software is used to convert decimal numbers into binary numbers, and vice versa.

Number Systems

Binary: 0, 1

Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Octal: 0, 1, 2, 3, 4, 5, 6, 7

Number Systems

- Binary numbers tend to be very long and cumbersome:
 - For example: 1010 1010 1010
- Hexadecimal numbers are often used to abbreviate binary numbers:
 - For example: AAA
- The hexadecimal number system has 16 digits:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
 - The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15.
- The octal number system has 8 digits:
 - 0, 1, 2, 3, 4, 5, 6, and 7
 - Each octal digit corresponds to 3 bits

Binary Numbers => Decimals

Given a binary number $b_nb_{n-1}b_{n-2}...b_2b_1b_0$
the equivalent decimal value is

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$10 \text{ in binary} \quad 1 \times 2^1 + 0 = 2 \text{ in decimal}$$

$$1010 \text{ in binary} \quad 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 = 10 \text{ in decimal}$$

$$\begin{array}{l} 10101011 \\ \text{in binary} \end{array} \quad 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 171 \text{ in decimal}$$

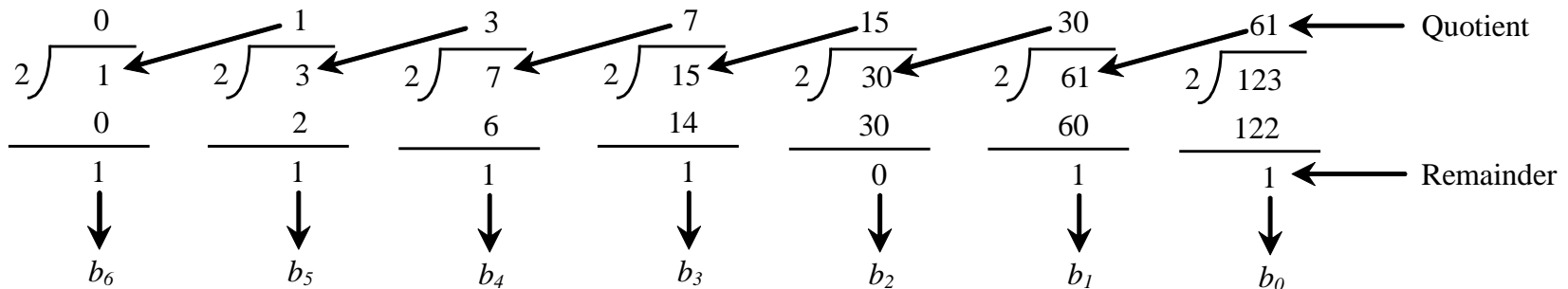
Decimals => Binary

- To convert a decimal number d to a binary number is to find the binary digits $b_n, b_{n-1}, b_{n-2}, \dots, b_2, b_1, b_0$ such that

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

- These numbers can be found by successively dividing d by 2 until the quotient is 0. The remainders are $b_n, b_{n-1}, b_{n-2}, \dots, b_2, b_1, b_0$

For example, the decimal number 123 is 1111011 in binary. The conversion is conducted as follows:



Hexadecimals \Leftrightarrow Binary

Binary	Hex	Decimal
--------	-----	---------

0000	0	0
------	---	---

0001	1	1
------	---	---

0010	2	2
------	---	---

0011	3	3
------	---	---

0100	4	4
------	---	---

0101	5	5
------	---	---

0110	6	6
------	---	---

0111	7	7
------	---	---

1000	8	8
------	---	---

1001	9	9
------	---	---

1010	A	10
------	---	----

1011	B	11
------	---	----

1100	C	12
------	---	----

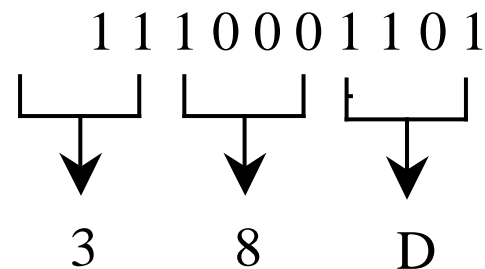
1101	D	13
------	---	----

1110	E	14
------	---	----

1111	F	15
------	---	----

To convert a hexadecimal number to a binary number, simply convert each digit in the hexadecimal number into a **four-digit** binary number.

To convert a binary number to a hexadecimal, convert every **four binary digits from right to left** in the binary number into a hexadecimal number. For example,



Hexadecimals => Decimals

- The hexadecimal number system has sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
- The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15.
- Given a hexadecimal number $h_n h_{n-1} h_{n-2} \dots h_2 h_1 h_0$

The equivalent decimal value is

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

$$7F \text{ in hex is } 7 \times 16^1 + 15 = 127 \text{ in decimal}$$

$$FFFF \text{ in hex } 15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15 = 65535 \text{ in decimal}$$

- Octal number system is similar, but *base* is 8.

Decimals \Rightarrow Hexadecimals

To convert a decimal number d to a hexadecimal number is to find the hexadecimal digits $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1, h_0$ such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These numbers can be found by successively dividing d by 16 until the quotient is 0. The remainders are

$$h_0, h_1, h_2, \dots, h_{n-2}, h_{n-1}, h_n$$

For example, the decimal number 123 is 7B in hexadecimal. The conversion is conducted as follows:

$$\begin{array}{r} 0 \\ 16 \overline{) 7} \\ \underline{0} \\ 7 \end{array} \quad \begin{array}{r} 7 \\ 16 \overline{) 123} \\ \underline{112} \\ 11 \end{array}$$

Quotient

Remainder

\downarrow \downarrow

h_1 h_0

Octal number system is similar, but *base* is 8.

Octal \Leftrightarrow Binary

Binary Octal Decimal

000 0 0

001 1 1

010 2 2

011 3 3

100 4 4

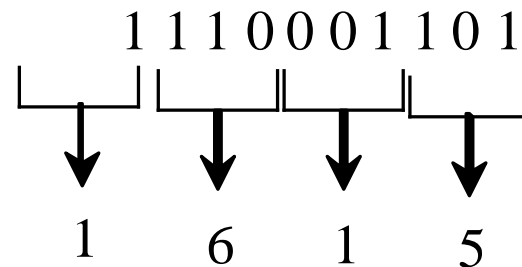
101 5 5

110 6 6

111 7 7

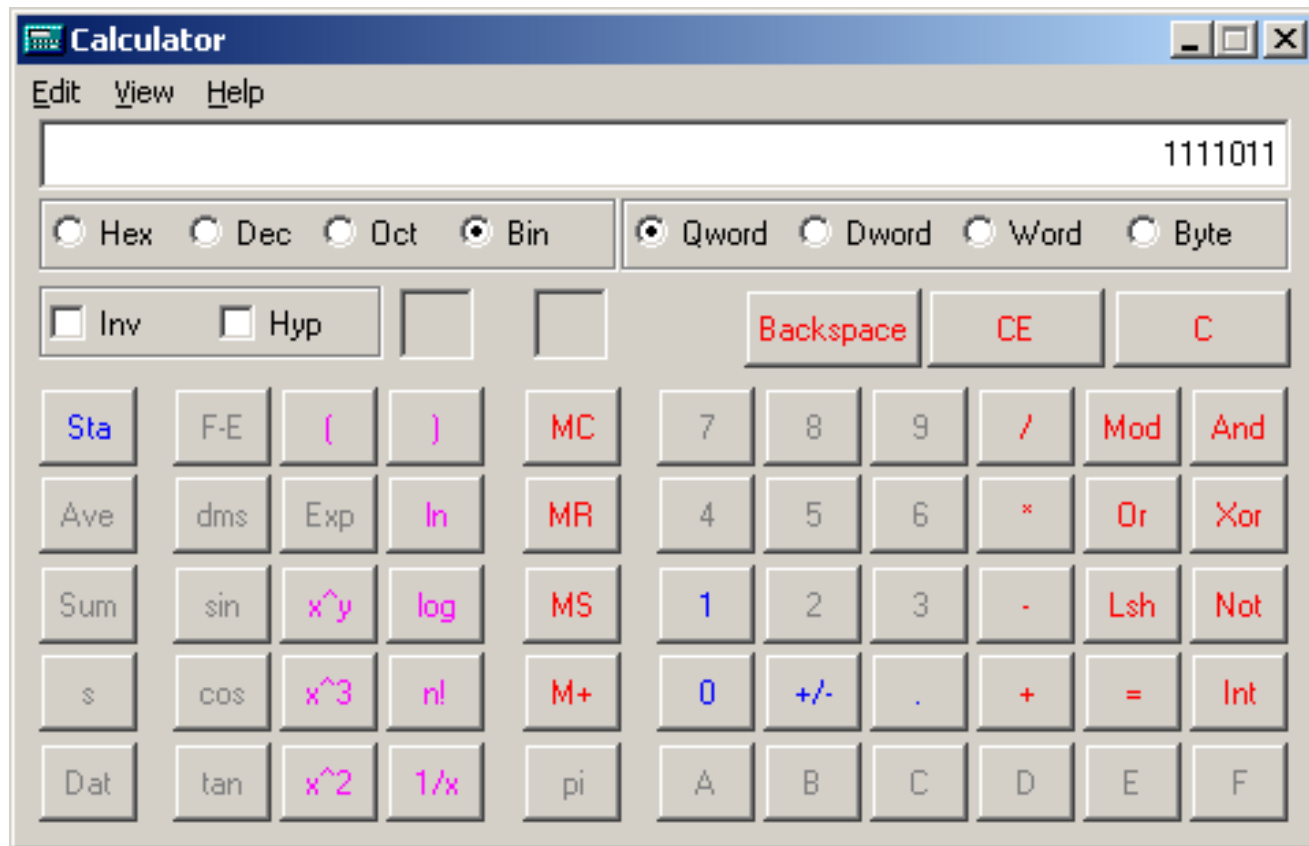
To convert an octal number to a binary number, simply convert each digit in the octal number into a **three-digit** binary number.

To convert a binary number to an octal number, convert every **three binary digits from right to left** in the binary number into an octal digit. For example,



Windows Calculator

The Windows Calculator is a useful tool for performing number conversions. To run it, choose *Programs*, *Accessories*, and *Calculator* from the Start button.



Memory: What goes in each memory segment?

- **Stack Segment**

- temporary variables declared inside methods
- removed from memory when a method returns

- **Heap Segment**

- for dynamic data (whenever you use new)
- data for constructed objects
- persistent as long as an existing object variable references this region of memory

- **Global Segment**

- data that can be reserved at compile time
- global data (like static data)



Stack Segment

Heap Segment

Global Segment

How objects are stored?

- You must understand that in Java, every object/reference variable stores a memory address
 - 32 bit numbers (4 bytes)

OR

- 64 bit numbers (8 bytes)
- These addresses point to memory locations where the objects' data is stored

So Hardware stores 0s & 1s

- 01010101010101010101010101010101 ...
- Data is byte addressable
 - we can access or change any byte (group of 8 bits) independently as needed
- **How do we store text?**
 - Numerically (using its code)
 - Each character is stored in memory as a number
 - Standard character sets: ASCII & Unicode
 - ASCII uses 1 byte per character
 - 'A' is 65

ASCII Table

http://enteos2.area.trieste.it/russo/IntroInfo2001-2002/CorsoRetiGomez/ASCII-EBIC_files/ascii_table.jpg

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Programming Languages

Machine Language Assembly Language High-Level Language

- Machine language is a set of instructions executed directly by a computer's central processing unit (CPU).
 - At the beginning there was only machine language: a sequence of bits that directly controls a processor, causing it to add, compare, move data from one place to another
 - Example: GCD program in x86 machine language:

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```

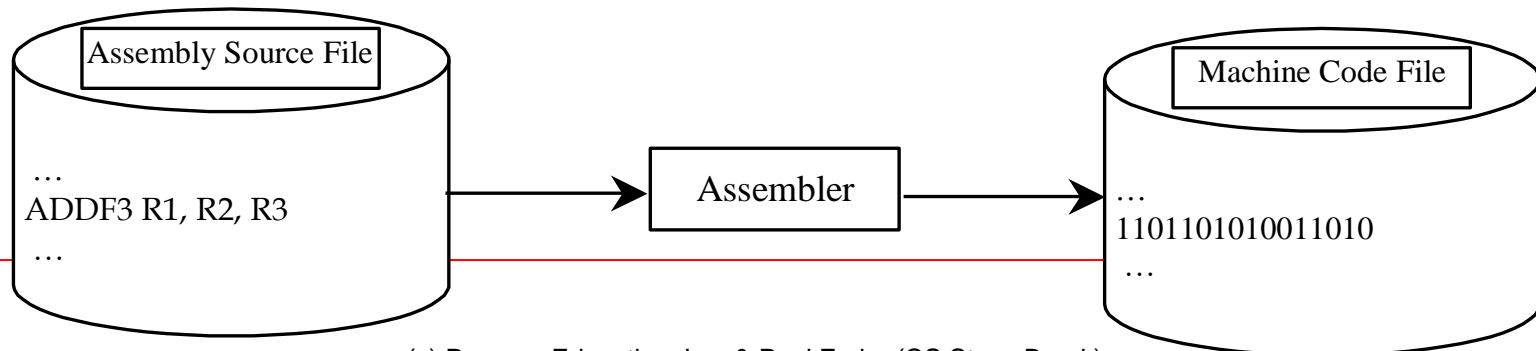
Programming Languages

Machine Language Assembly Language High-Level Language

- *Assembly languages* were invented to allow operations to be expressed with **mnemonic** abbreviations
- A program called **assembler** is used to convert assembly language programs into machine code

For example, to add two numbers, you might write an instruction in assembly code like this:

ADDF3 R1, R2, R3



Programming Languages

Machine Language Assembly Language High-Level Language

- *Example: GCD program in x86 assembly:*

```
    pushl    %ebp
    movl     %esp, %ebp
    pushl    %ebx
    subl     $4, %esp
    andl     $-16, %esp
    call     getint
    movl     %eax, %ebx
    call     getint
    cmpl     %eax, %ebx
    je       C
A:   cmpl     %eax, %ebx
    jle     D
    subl     %eax, %ebx
B:   cmpl     %eax, %ebx
    jne     A
C:   movl     %ebx, (%esp)
    call     putint
    movl     -4(%ebp), %ebx
    leave
    ret
D:   subl     %ebx, %eax
    jmp     B
```

Programming Languages

Machine Language Assembly Language High-Level Language

Assembly: Far easier to use than binary machine learning
BUT: not very user friendly, very low-level operations,
programming is time consuming

High Level programming Languages (HLL):

- more user friendly, easy to use
- more flexible
- platform independent

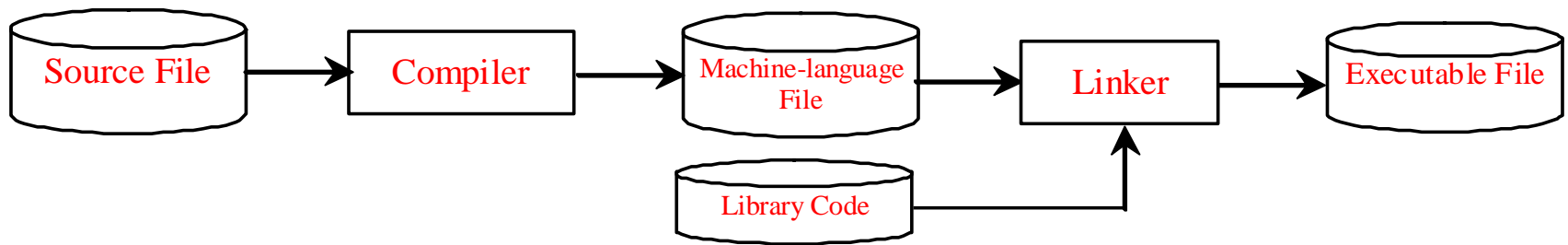
Popular High-Level Languages

- FORTRAN (FORmula TRANslation)
- LISP
- COBOL (COmmon Business Oriented Language)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (named for Blaise Pascal)
- Ada (named for Ada Lovelace)
- C (whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)
- Java
- C# (a Java-like language developed by Microsoft)
- python

Compiling Source Code

What's a compiler?

- A software program
 - Input: High Level Language source code
 - Output: Assembly Code
- It is typically integrated with an assembler
 - together they can make an executable or binary program



Operating Systems

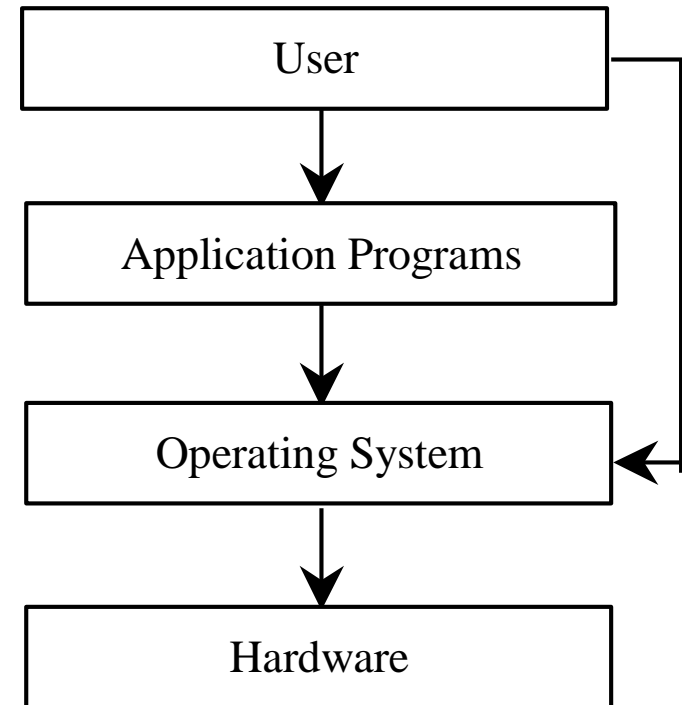
The operating system (OS) is a program that manages and controls a computer's activities

Windows

Mac OsX

Android

Linux



Why Java?

Java is somewhat different from older languages

Java started a principle, “*write once, run anywhere*”

What does that mean?

Platform independence for compiled Java code

How?

The Java Virtual Machine

Java programs are compiled into Java bytecode

Bytecode is then executed by the

Java Virtual Machine (JVM)

Java, JVM, Web, and Beyond

- **Java Virtual Machine**

- A program that runs Java programs and manages memory for Java programs.

- **Why?**

- Each platform is different (Mac/PC/Linux/etc.)
- Java can be used to develop Web applications.
- Java Applets
- Java Web Applications
- Java can also be used to develop applications for hand-held devices such as Palm and cell phones

JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- J2SE 1.2 (1998)
- J2SE 1.3 (2000)
- J2SE 1.4 (2002)
- J2SE 5.0 (2004)
- Java SE 6 (2006)
- Java SE 7 (2011)
- Java SE 8 (2014) Long Term Support (LTS)
- Java SE 9 (2017)
- Java SE 10, 11 (LTS) (2018)
- Java SE 12, 13 (2019)
- Java SE 14 (expected March 2020)

JDK Editions

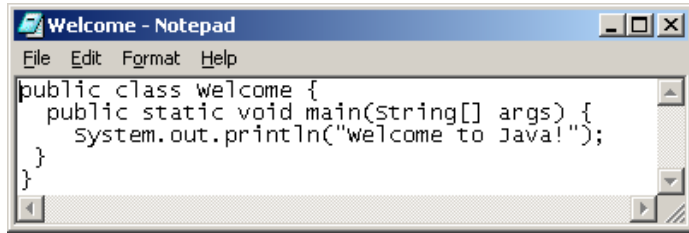
- Java Standard Edition (J2SE)
 - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
 - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME) .
 - J2ME was used to develop applications for mobile devices such as cell phones.

Our textbook uses J2SE to introduce Java programming.

A Simple Java Program

```
//    Welcome.java
//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```


Creating, Compiling, and Running Programs

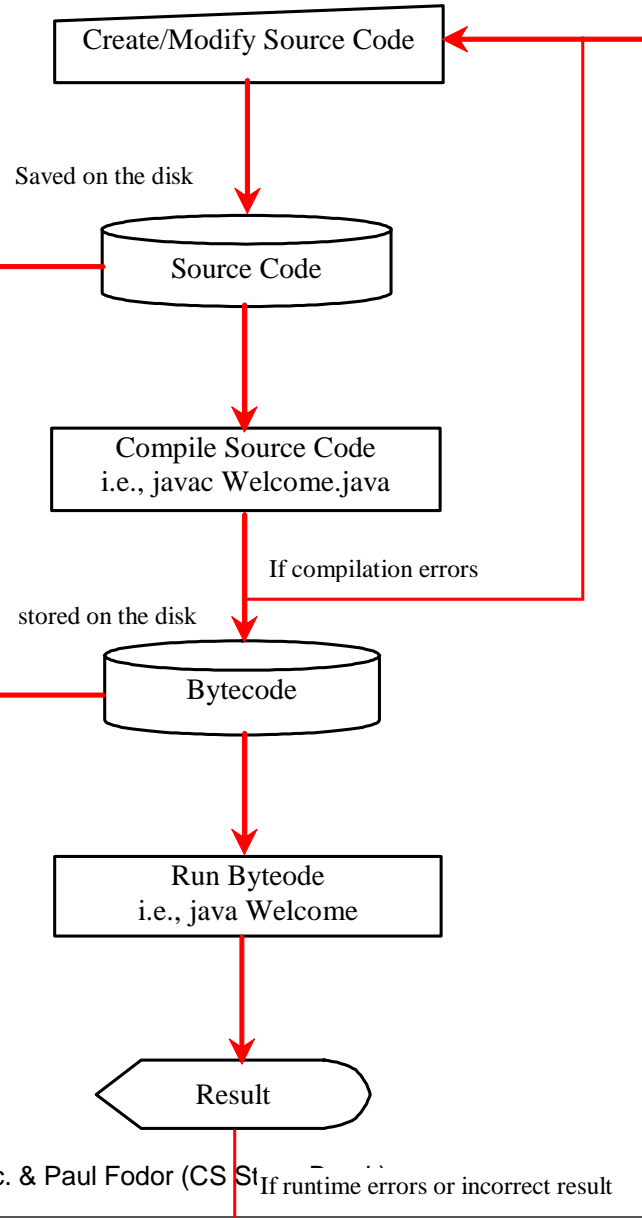


Source code (developed by the programmer)

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

Byte code (generated by the compiler for JVM to read and interpret, not for you to understand)

```
...  
Method Welcome()  
  0 aload_0  
  ...  
Method void main(java.lang.String[])  
  0 getstatic #2 ...  
  3 ldc #3 <String "Welcome to Java!">  
  5 invokevirtual #4 ...  
  6 return
```



Running Programs from command line

```
pfodor@sparky ~$ emacs Welcome.java
```

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

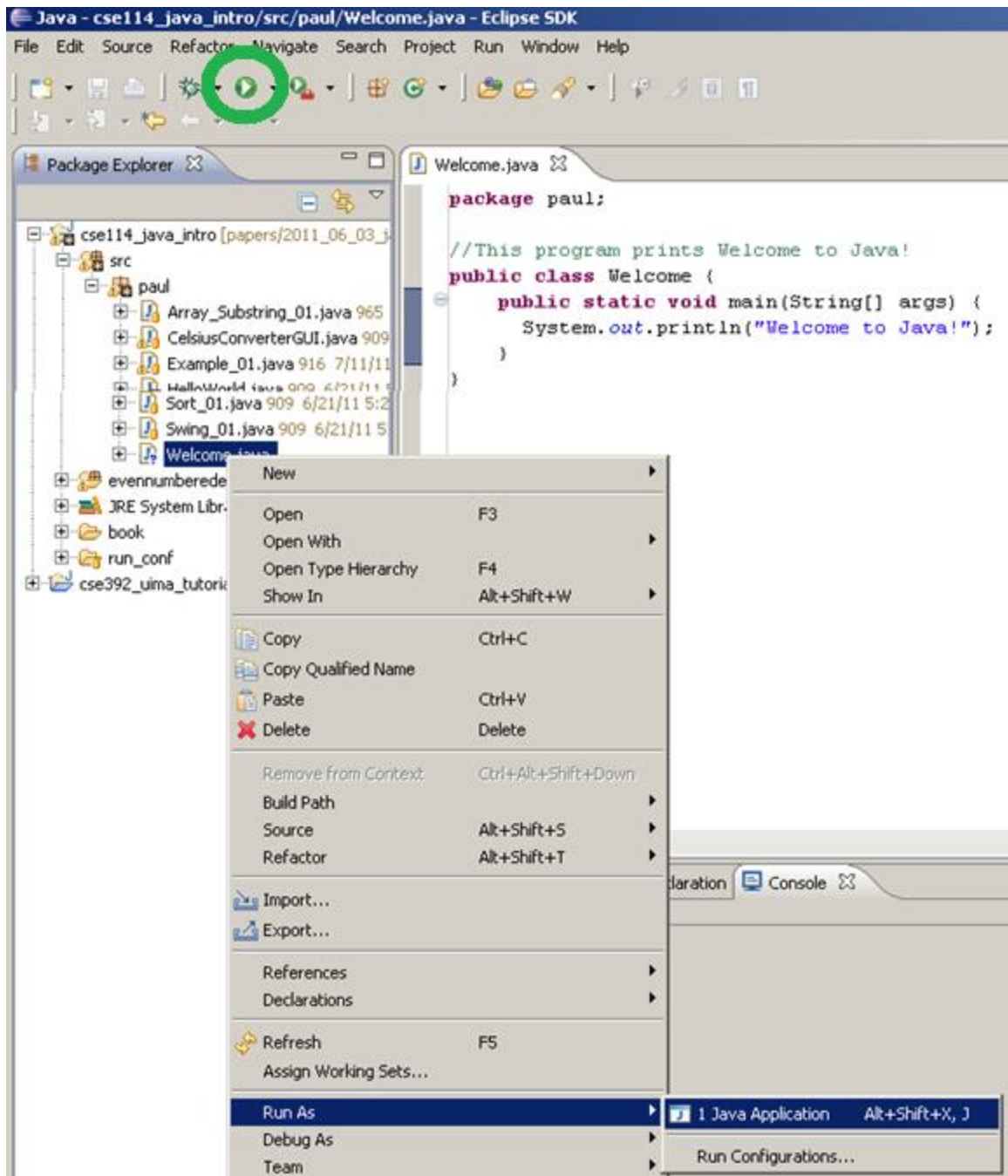
```
pfodor@sparky ~$ javac Welcome.java
```

```
pfodor@sparky ~$ java Welcome  
Welcome to Java!
```

Compiling and Running Java from the Command Window

- Set path to JDK bin directory
set PATH=c:\Java\jdk1.8.0\bin
- Set classpath to include the current directory
set CLASSPATH=.
- Compile your source code:
javac Welcome.java
- Run your bytecode:
java Welcome

Running Programs in Eclipse



Trace a Program Execution

Enter main method

```
//This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

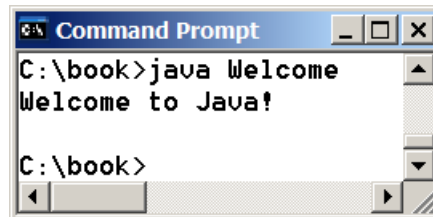
Trace a Program Execution

Execute statement

```
//This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

Trace a Program Execution

```
//This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



print a message to the console

Anatomy of a Java Program

- Comments
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

- Three types of comments in Java.

Line comment: A line comment is preceded by two slashes (//) in a line.

Paragraph comment: A paragraph comment is enclosed between /* and */ in one or multiple lines.

javadoc comment: javadoc comments begin with /** and end with */. They are used for documenting classes, data, and methods. They can be extracted into an HTML file using JDK's javadoc command.

Reserved Words (Keywords)

- *Reserved words* or *keywords* are words that have a specific meaning to the compiler
 - Cannot be used for other purposes in the program
 - Example: **class**
 - the word after **class** is the name for the class

Java Keywords

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

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>

Modifiers

- Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used
 - Examples: **public**, **static**, **private**, **final**, **abstract**, **protected**
 - A **public** datum, method, or class can be accessed by other programs
 - A **private** datum or method cannot be accessed by other programs

Statements

- A statement represents an action or a sequence of actions

```
System.out.println("Welcome to Java!");
```

is a statement to display the greeting "Welcome to Java!"

- Every statement in Java ends with a semicolon (;)

Blocks

A pair of braces in a program forms a block that groups components of a program.

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

Class block

Method block

Block Styles

- We use end-of-line style for braces:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("Block Styles");  
    }  
}
```

*End-of-line
style*

*Next-line
style*

```
public class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Block Styles");  
    }  
}
```

Variable, class, and method names

- What's an API?
 - Application Programming Interface
 - a library of code to use
 - Names
 - For Variables, Classes, and Methods
 - From 2 sources:
 - your own classes, variables, and methods
 - the Oracle/Sun (or someone else's) API
 - Your Identifiers (Names) – Why name them?
 - they are your data and commands
 - you'll need to reference them elsewhere in your program
- ```
int myVariable = 5; // Declaration
myVariable = myVariable + 1; // Using the variable
```



# Rules for Identifiers

- Should contain only letters, numbers, & '\_'
  - '\$' is allowed, but only for special use
- Cannot begin with a digit!
- Uppercase and lowercase letters are considered to be different characters
- Examples:
  - Legal: **myVariable, my\_class, my4Var**
  - Illegal: **4myVariable, my class, my!Var, @\$myClass**

# Common Java Naming Conventions

- Variables & Methods start with lower case letters: **x**, **toString**
- Classes start with upper case letters: **Person**
- Variables and Class identifiers should generally be nouns
- Method identifiers should be verbs
- Use Camel notation: **myVariable**, **MyClass**
- Although it is legal, do not begin with ‘\_’ (underscore).
- Use descriptive names: **LinkedList**, **compareTo**

```
area = PI * radius * radius;
```

# Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

# Syntax Error

```
public class ShowSyntaxError {
 public static void main(String[] args) {
 i = 30; // Detected by the compiler
 System.out.println(i + 4);
 }
}
```

# Runtime Error

```
public class ShowRuntimeError {
 public static void main(String[] args) {
 int i = 1 / 0; // Division with 0
 }
}
```

# Logic Errors

```
public class ShowLogicError {
 // Determine if a number is between 1 and 100 inclusively
 public static void main(String[] args) {
 Scanner input = new Scanner(System.in);
 int number = input.nextInt();
 // Display the result
 System.out.println(
 "The number is between 1 and 100, inclusively: " +
 ((1 < number) && (number < 100)));
 // Wrong result if the entered number is 1 or 100
 System.exit(0);
 }
}
```

# Logic Errors Debugging

- Logic errors are also called bugs
- The process of finding and correcting errors is called debugging
- Methods:
  - hand-trace the program (i.e., catch errors by reading the program),
  - insert print statements in order to show the values of the variables
  - for a large, complex program, the most effective approach for debugging is to use a debugger utility

# Debugger

Debugger is a program that facilitates debugging. You can use a debugger to:

- Set breakpoints.
- Execute a single statement at a time.
- Trace into or stepping over a method.
- Display variables.
- Display call stack.
- Modify variables.



Package Explorer

- \_\_CSE114 [\_\_CSE114]
- \_\_CSE114\_2013\_Fall [\_\_CSE114\_2013\_Fall]
- \_\_CSE114\_2013\_Spring [\_\_CSE114\_2013\_Spring]
- edu.sunysb.bench\_svn [edu.sunysb.bench]
- RuleMLPaper [RuleMLPaper]
- sbnlp [svn://ewl.cewit.stonybrook.edu/home/pfodor/repositories/svn]
- sbpl [svn://ewl.cewit.stonybrook.edu/home/pfodor/sbpl]
- sbsvn [svn://ewl.cewit.stonybrook.edu/sbsvn]
- svn\_repository\_vikas [svn://ewl.cewit.stonybrook.edu/home/pfodor/svn\_repository\_vikas]
- WPLE\_svn [svn://ewl.cewit.stonybrook.edu/WPLE\_svn]
- xsb\_eclipse [trunk/xsb\_eclipse]

```
public void refreshDisplay(String option){
 System.out.println("Option: "+option);
 if(option.equals("b41") || option.equals("b42")){
 l1.setText("Round "+rounds+" "+"Act
 if(option.equals("b41")){
 bet = 1;
 }else if(option.equals("b42")){
 bet = 2;
 }else if(option.equals("b43")){
 bet = 3;
 }else if(option.equals("b44")){
 bet = 4;
 }else bet = 5;
 l42.setText("Bet: $" + bet);
 b41.disable();
 b42.disable();
 b51.enable();
 b6.disable();
 } else if(option.equals("b52")){
 // implement second step of baccarat
 }
```

Debug

Baccarat\_GUI\_draft [Java Application]

- Baccarat\_GUI\_draft at localhost:3008
  - Thread [AWT-Shutdown] (Running)
  - Daemon Thread [AWT-Windows] (Running)
  - Thread [AWT-EventQueue-0] (Suspended (breakpoint at line 159 in Baccarat\_GUI\_draft))
    - Baccarat\_GUI\_draft.refreshDisplay(String) line: 159
    - Baccarat\_GUI\_draft.actionPerformed(ActionEvent) line: 101
    - JButton(AbstractButton).fireActionPerformed(ActionEvent) line: not available
    - AbstractButton\$Handler.actionPerformed(ActionEvent) line: not available
    - DefaultButtonModel.fireActionPerformed(ActionEvent) line: not available
    - DefaultButtonModel.setPressed(boolean) line: not available
    - BasicButtonListener.mouseReleased(MouseEvent) line: not available
    - JButton(Component).processMouseEvent(MouseEvent) line: not available
    - JButton(JComponent).processMouseEvent(MouseEvent) line: not available
    - JButton(Component).processEvent(AWTEvent) line: not available
    - JButton(Container).processEvent(AWTEvent) line: not available

Variables Breakpoints

| Name                    | Value                   |
|-------------------------|-------------------------|
| b41                     | JButton (id=72)         |
| b42                     | JButton (id=73)         |
| b43                     | JButton (id=74)         |
| b44                     | JButton (id=77)         |
| b45                     | JButton (id=60)         |
| b51                     | JButton (id=78)         |
| b6                      | JButton (id=79)         |
| background              | ColorUIResource (id=80) |
| backgroundEraseDisabled | false                   |

Baccarat\_GUI\_draft.java

```

System.out.println("Option: "+option);
if(option.equals("b41") || option.equ
 l1.setText("Round "+rounds+"
 if(option.equals("b41")){
 bet = 1;
 }else if(option.equals("b42")){

```

Outline

```

new ActionListener() {...}
main(String[]) : void
refreshDisplay(String) : void
card_to_ImageIcon(int[]) : ImageIcon
sum_hand(int[], int) : int
generate_card(int[], int) : int[]
generate_card() : int[]
card_to_String(int[]) : String

```

Console Tasks

Baccarat\_GUI\_draft [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Sep 12, 2013 11:17:36 AM)