Java Programming

Dr. Ferdin Joe John Joseph
Kamnoetvidya Science Academy

Java Vs. Java Script

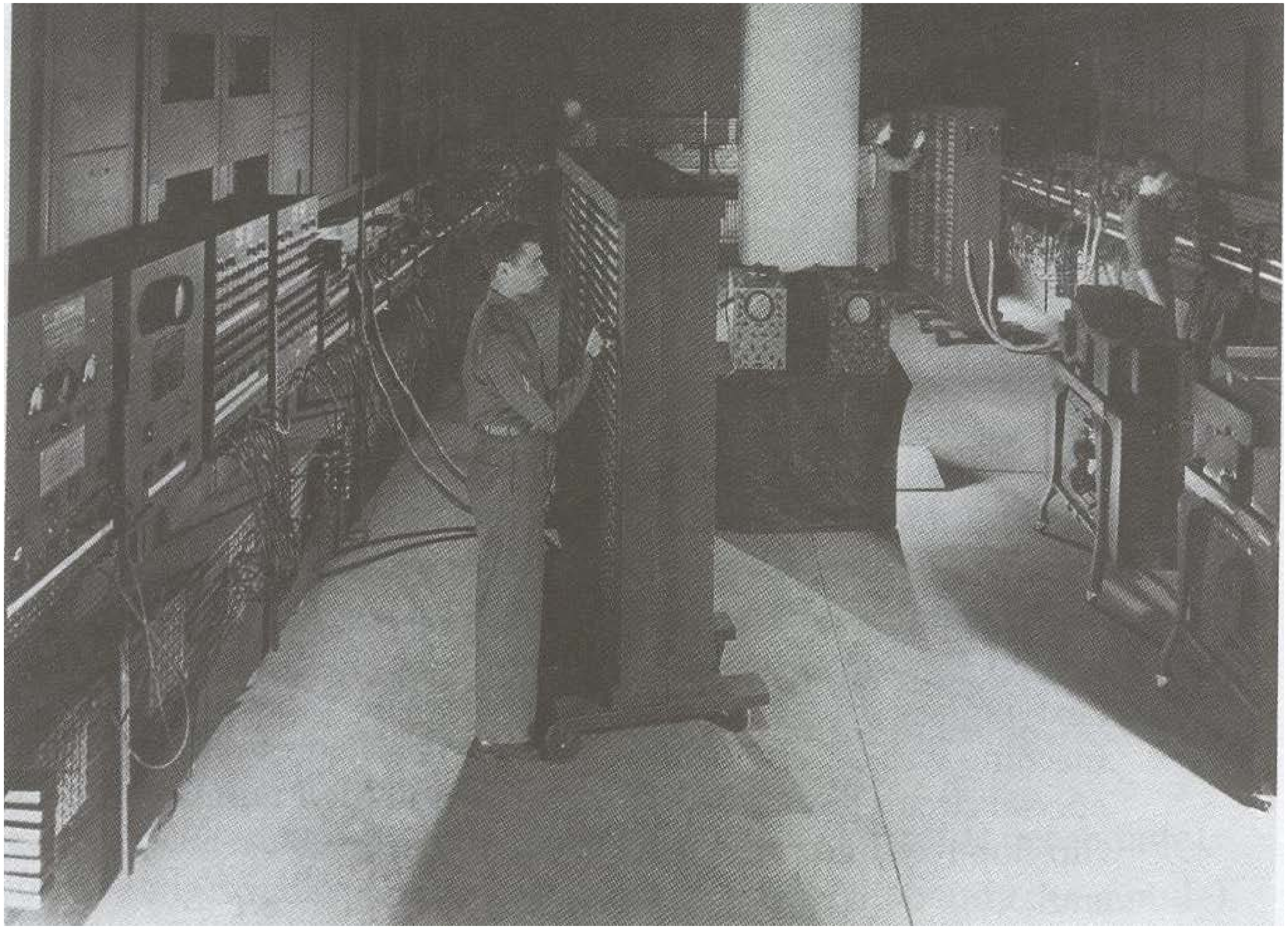
Java (*this is what you need to know for this course*)

- A complete programming language developed by Sun
- Can be used to develop either web based or stand-alone software
- Many pre-created code libraries available
- For more complex and powerful programs

Java Script (*not covered in this course*)

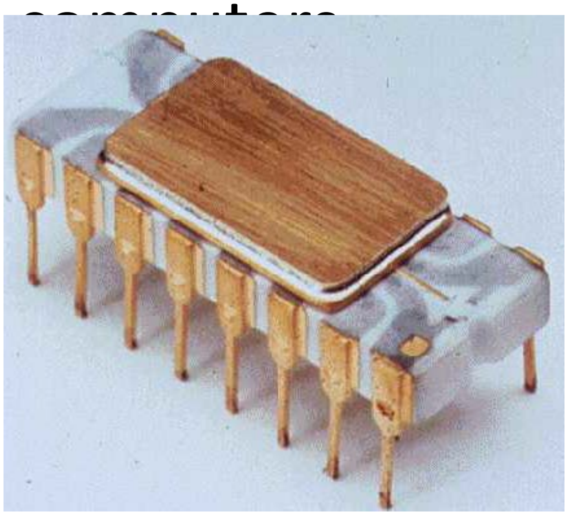
- A small language that's mostly used for web-based applications (run through a web browser like Internet Explorer, Firefox, Safari, Chrome)
- Good for programming simple special effects for your web page e.g., roll-overs

Java: History



Java: History (2)

- The invention of the microprocessor revolutionized computers


Intel microprocessor

Commodore Pet microcomputer

Java: History (3)

Java History (4)

- Sun Microsystems funded an internal research project "Green" to investigate this opportunity.
  - Result: A programming language called "Oak"

Blatant advertisement: James Gosling was a graduate of the U of C Computer Science program.

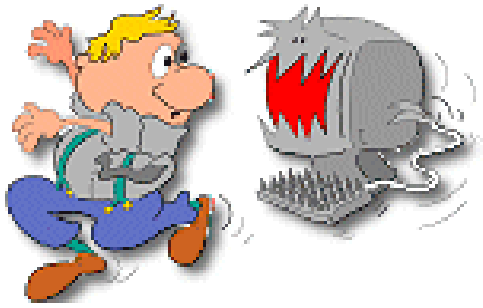
Java History (5)

- Problem: There was already a programming language called Oak.
- The "Green" team met at a local coffee shop to come up with another name...

JAVA™

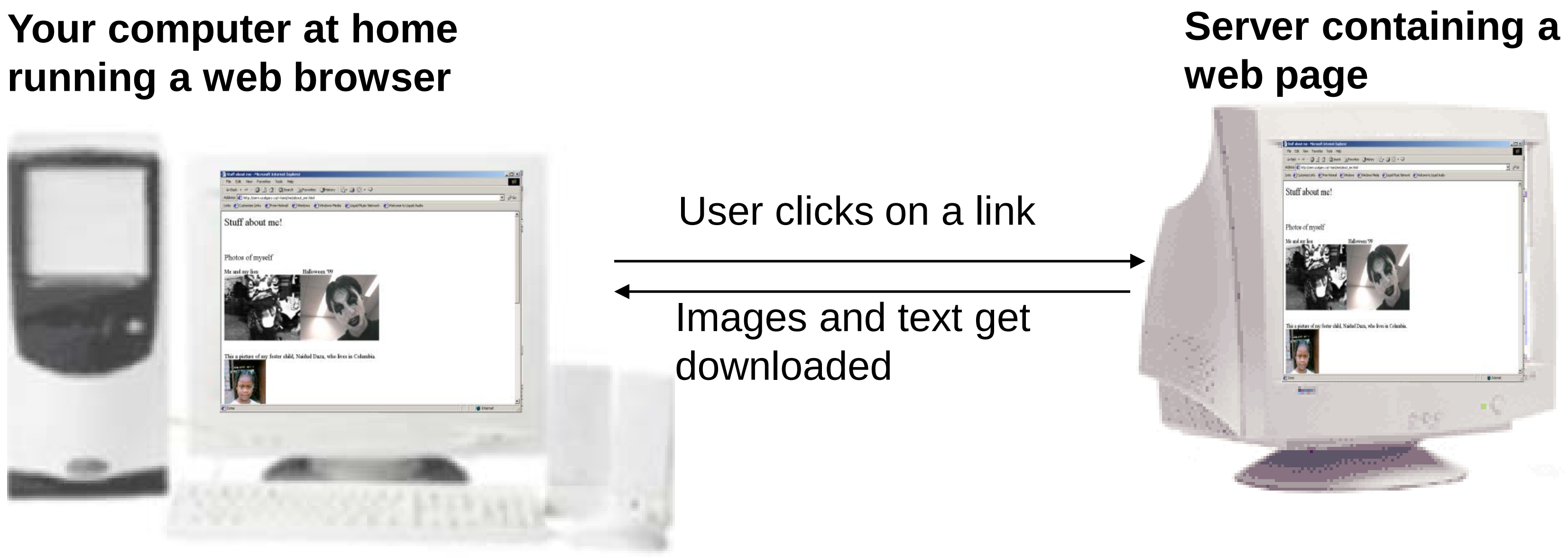
Java: History (6)

- The concept of intelligent devices didn't catch on.
- Project Green and work on the Java language was nearly canceled.



Java: History (7)

- The popularity of the Internet resulted in Sun's re-focusing of Java on computers.
- Prior to the advent of Java, web pages allowed you to download only text and images.


Your computer at home running a web browser
Server containing a web page
User clicks on a link
Images and text get downloaded

Java: Write Once, Run Anywhere

- Consequence of Java's history: platform-independence

Click on link to Applet

Mac user running Safari

Virtual machine translates byte code to native Mac code and the Applet is run

Taipei Times - photo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://taipeitimes.com/News/biz/photo/2003/08/07/0000060471
TAIPEI TIMES
Thu, Aug 07, 2003
News Editorials Industry
Front Page
Taiwan News
World News
Editorials
Sports
Business
World Business
Features
Photo News
MasterCard
Goodbye Kitty

Web page stored on Unix server

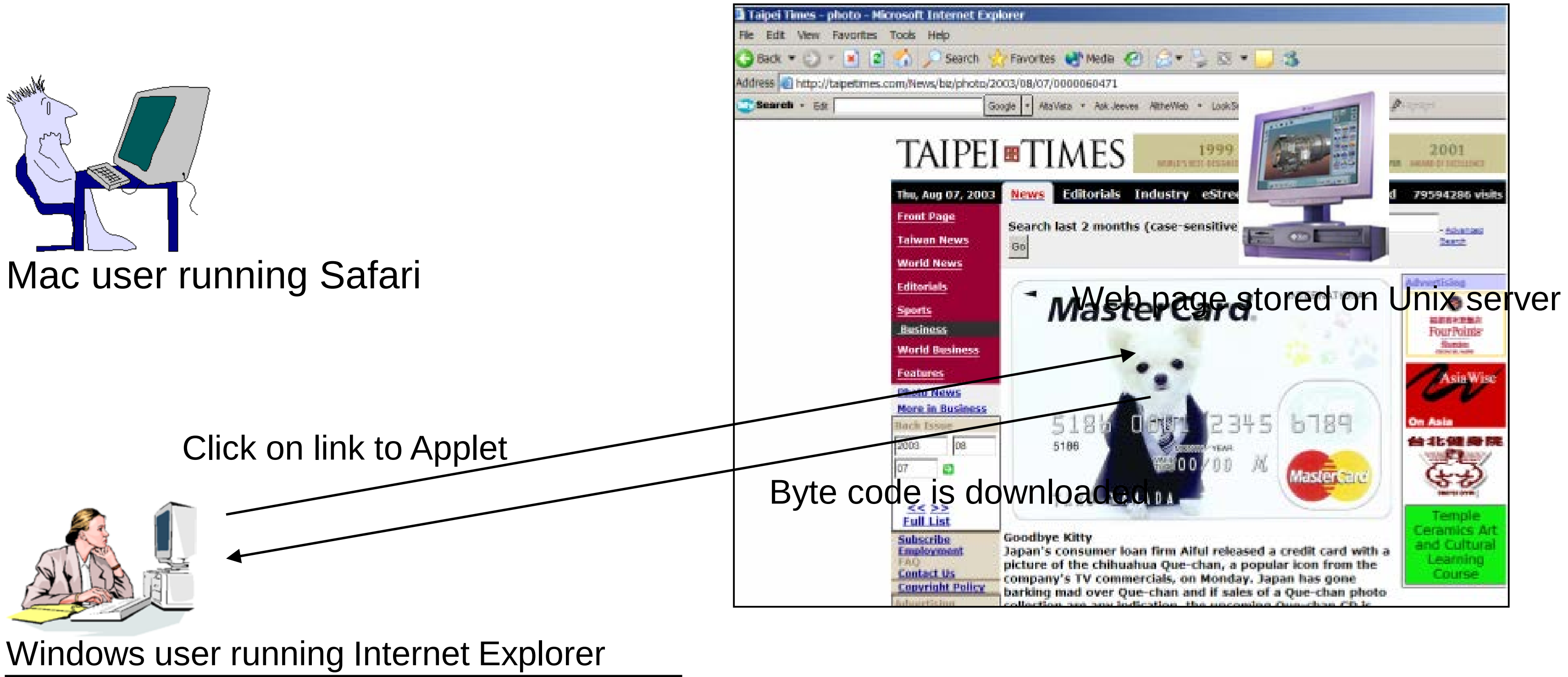
Byte code is downloaded

Windows user running Internet Explorer

Byte code (part of web page)

Java: Write Once, Run Anywhere

- Consequence of Java's history: platform-independent



Virtual machine translates byte code to native Windows code and the Applet is run

Java: Write Once, Run Anywhere (2)

- But Java can also create standard (non-web based)

Dungeon Master (Java version)
http://homepage.mac.com/aberfield/dmj/

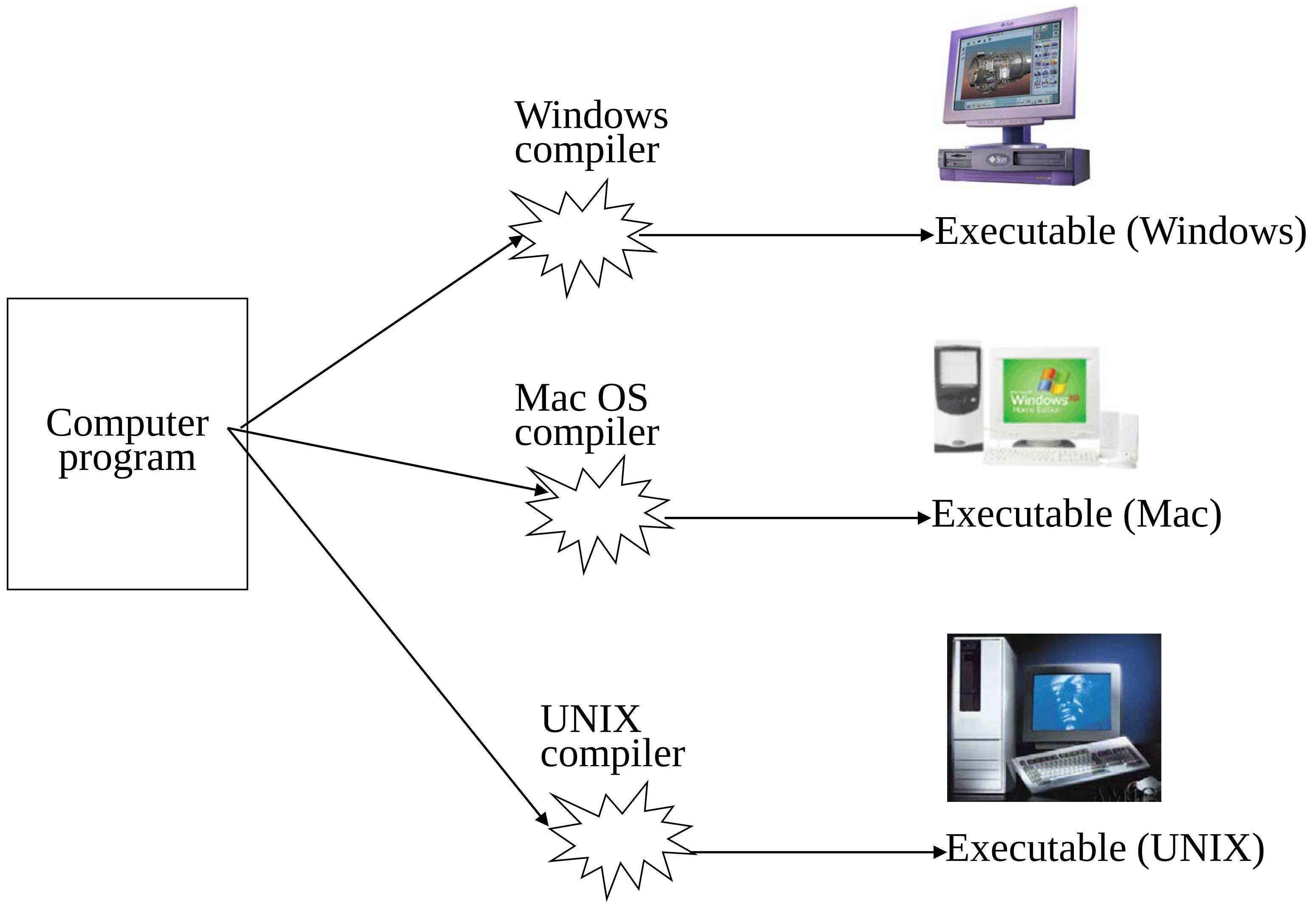
Kung Fu Panda 2: THQ

Java: Write Once, Run Anywhere (3)

- Java has been used by large and reputable companies to create serious stand-alone applications.
- Example:
  - Eclipse[1]: started as a programming environment created by IBM for developing Java programs. The program Eclipse was itself written in Java.

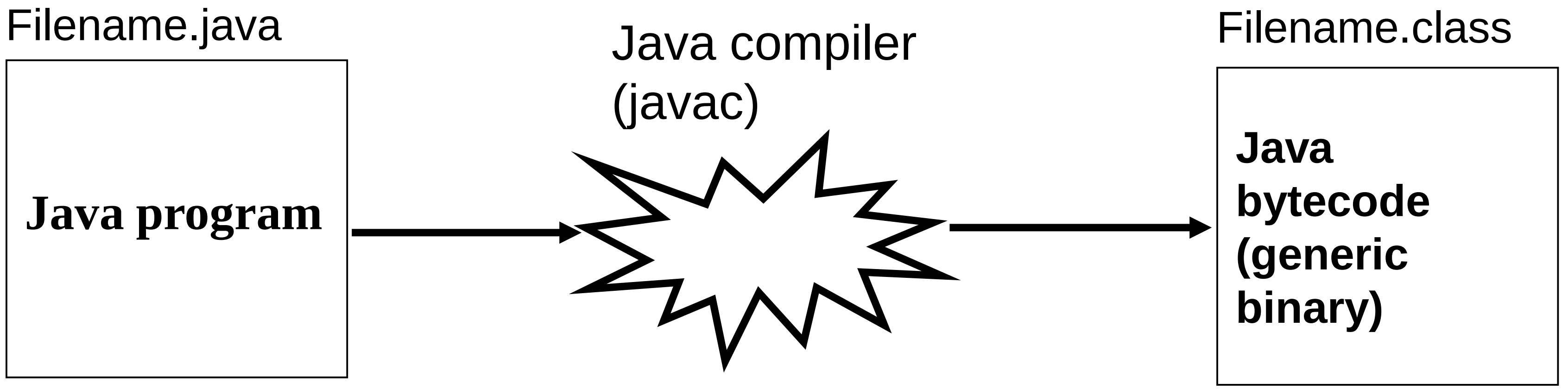
1 For more information: http://www.eclipse.org/downloads/

Compiled Programs With Different Operating Systems



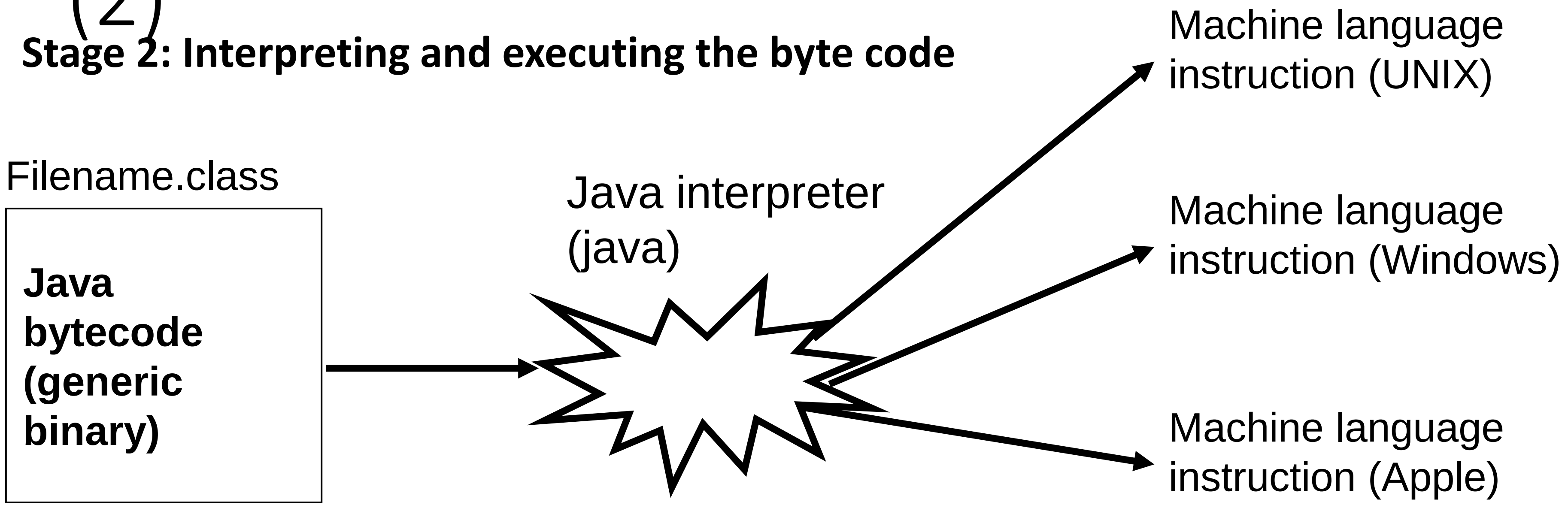
A High Level View Of Translating/Executing Java Programs

Stage 1: Compilation



A High Level View Of Translating/Executing Java Programs (2)

Stage 2: Interpreting and executing the byte code


Filename.class
Java bytecode (generic binary)
Java interpreter (java)
Machine language instruction (UNIX)
Machine language instruction (Windows)
Machine language instruction (Apple)

Which Java?

- Java 6+ JDK (Java Development Kit), Standard Edition includes:
  - J<u>*D*</u>K (Java development kit) – for *developing* Java software (creating Java programs.
  - J<u>*R*</u>E (Java Runtime environment) – only good for *running* pre-created Java programs.
    - Java Plug-in – a special version of the JRE designed to run through web browsers.

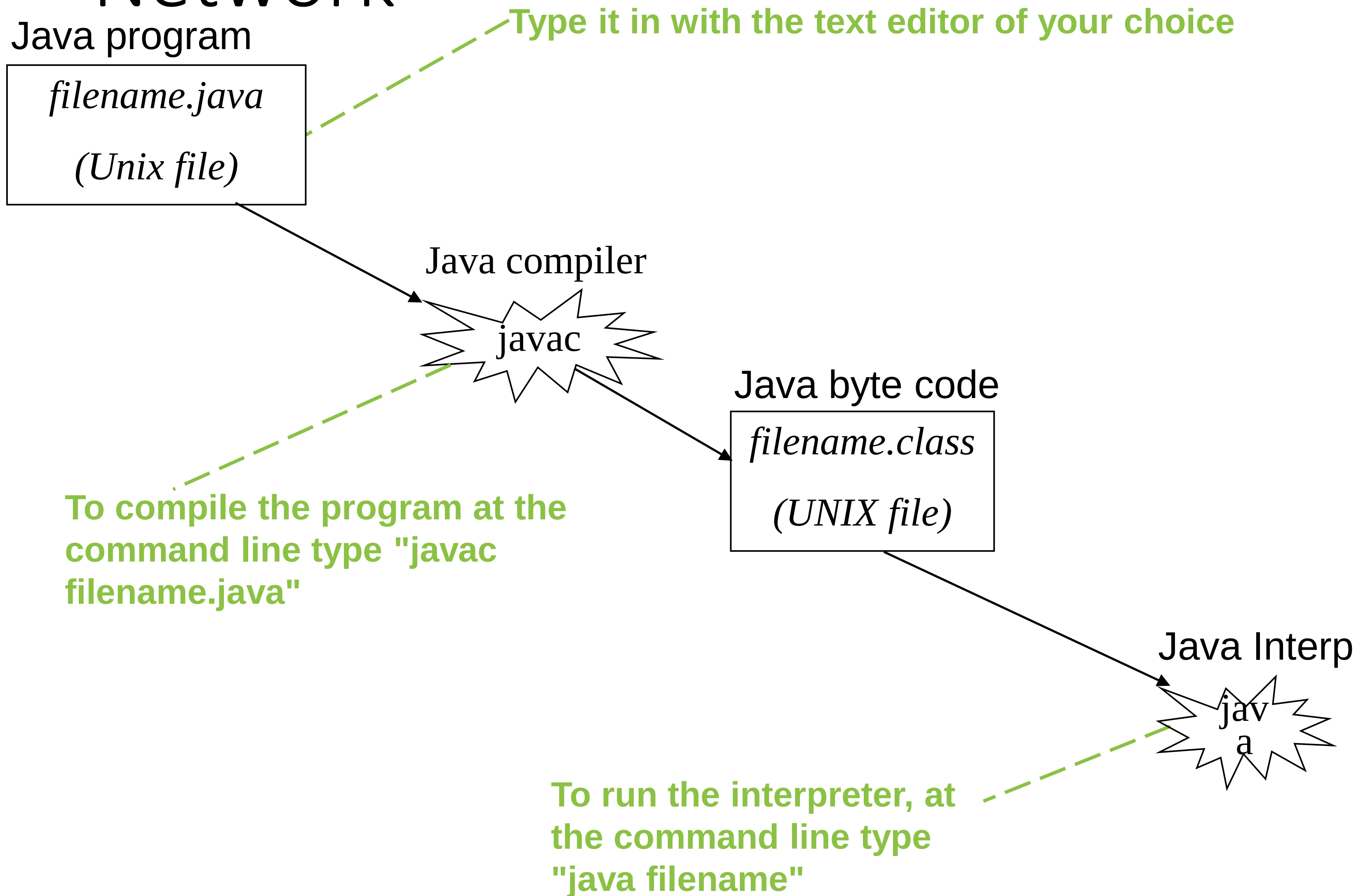
http://java.sun.com/javase/downloads/index.jsp

Smallest Compilable And Executable Java Program

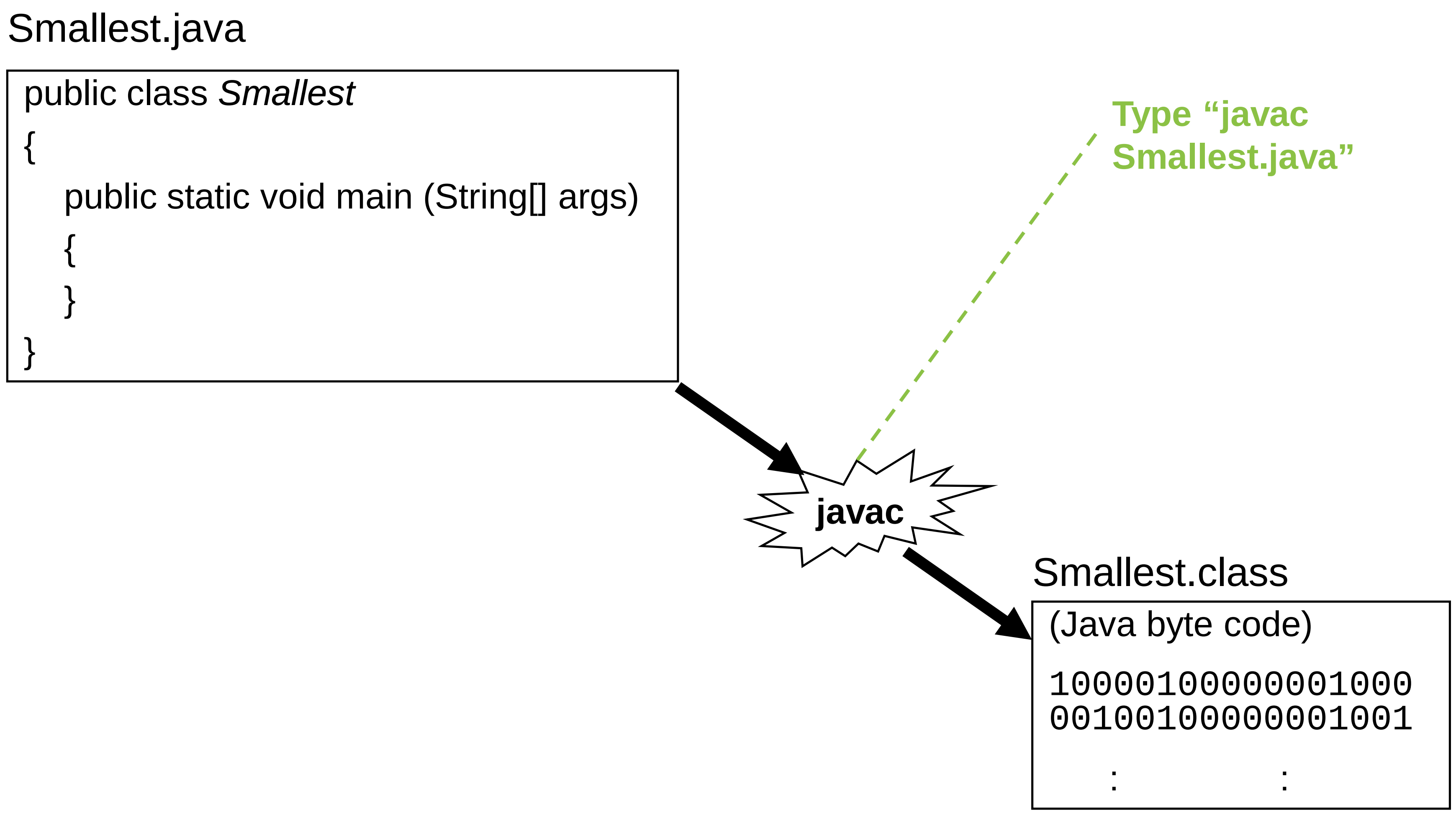
The name of the online example is: Smallest.java (*Important note: file name matches the word after the keyword ‘class’*)

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

Creating, Compiling And Running Java Programs On The Computer Science Network


Java program
Type it in with the text editor of your choice
filename.java
(Unix file)
Java compiler
javac
To compile the program at the command line type "javac filename.java"
Java byte code
filename.class
(UNIX file)
Java Interpreter
java
To run the interpreter, at the command line type "java filename"

Compiling The Smallest Java Program



Smallest.java

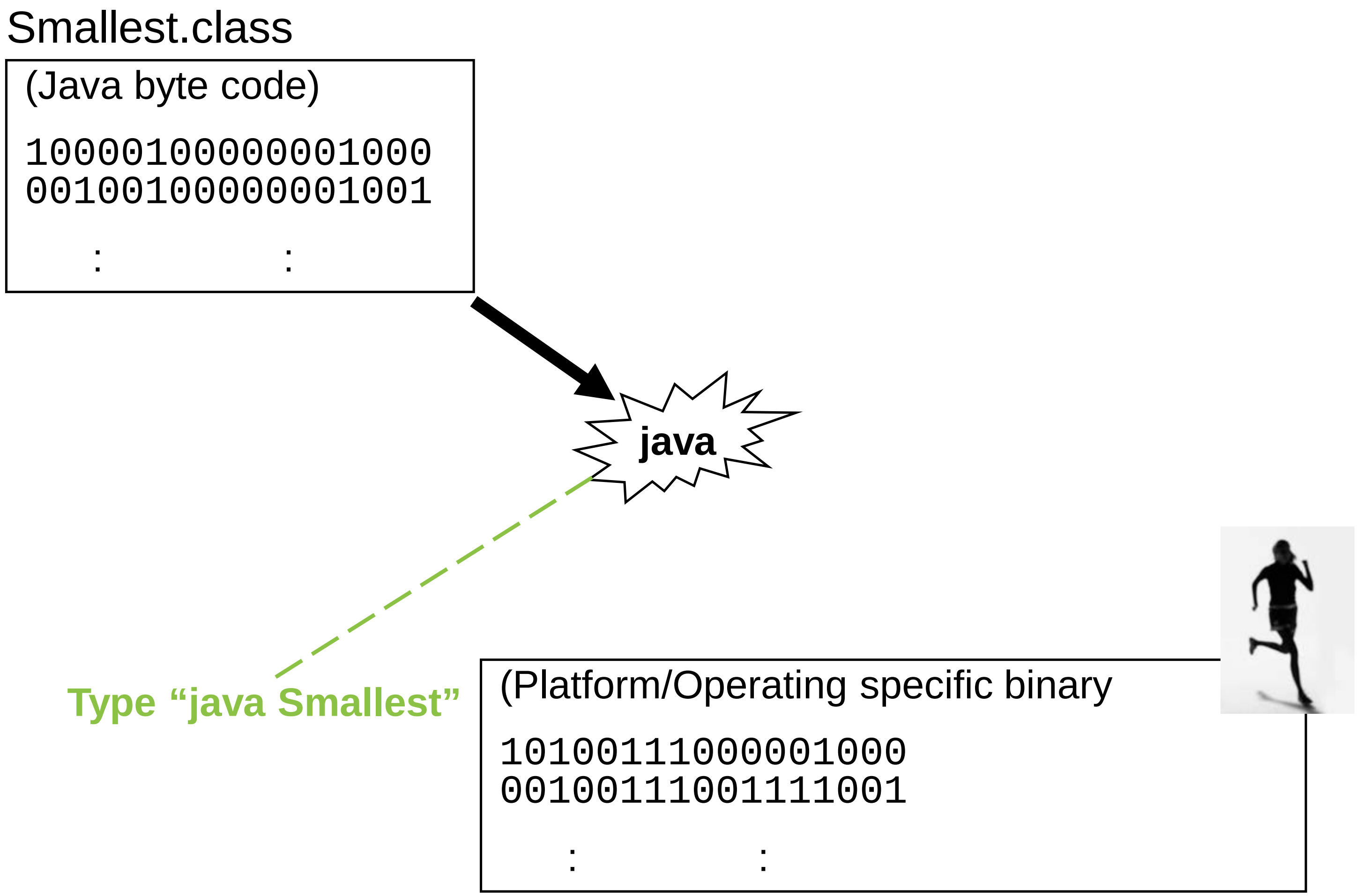
```
public class Smallest
{
    public static void main (String[] args)
    {
    }
}
```

Smallest.class

```
(Java byte code)

10000100000001000
00100100000001001
   :        :
```

Running The Smallest Java Program



(Java byte code)

```
1000010000001000
0010010000001001
    :        :
```

(Platform/Operating specific binary

```
1010011100001000
0010011100111001
    :        :
```

Documentation / Comments

Multi-line documentation

- /* Start of documentation
- */ End of documentation

Documentation for a single line

- //Everything until the end of the line is a comment

Review: What Should You Document

- Program (or that portion of the program) author
- What does the program as a while do e.g., tax program.
- What are the specific features of the program e.g., it calculates personal or small business tax.
- What are it's limitations e.g., it only follows Canadian tax laws and cannot be used in the US. In Canada it doesn't calculate taxes for organizations with yearly gross earnings over $1 billion.
- What is the version of the program
  - If you don't use numbers for the different versions of your program then consider using dates (tie versions with program features).

Important Note

- Each Java instruction must be followed by a semi-colon!

General format	**Examples**
Instruction1;	int num = 0;
Instruction2;	System.out.println(num);
Instruction3;	: :
: :	

Java Output

- **Format:**

  System.out.print(<*string or variable name one*> + <*string or variable name two*>..);
  OR
  System.out.println(<*string or variable name one*> + <*string or variable name two*>..);

- **Examples (online program called “OutputExample1.java”)**

```
public class OutputExample1
{
    public static void main (String [] args)
    {
        int num = 123;   // More on this shortly
          System.out.println("Good-night gracie!");
          System.out.print(num);
          System.out.println("num="+num);
     }
}
```

Output : Some Escape Sequences For Formatting

Escape sequence	**Description**
\t	Horizontal tab
\r	Carriage return
\n	New line
\”	Double quote
\\	Backslash

Example Formatting Codes

- Name of the online example: FormattingExample.java

```
public class FormattingExample
{
   public static void main (String [] args)
   {
      System.out.print("lol\tz\n");
      System.out.println("hello\rworld");
      System.out.println("\"Geek\" talk slash (\\) com");
   }
}
```

Variables

- Unlike Python variables must be declared before they can be used.
- Variable declaration:
  - Creates a variable in memory.
  - Specify the name of the variable as well as the type of information that it will store.
  - E.g. `int num;`
  - Although requiring variables to be explicitly declared appears to be an unnecessary chore it can actually be useful for minimizing insidious logic errors.
- Using variables
  - Only after a variable has been declared can it be used.
  - E.g., `num = 12;`

Declaring Variables: Syntax

- **Format**:

  <type of information> <name of variable>;

- **Example**:

```
char myFirstInitial;
```

- Variables can be initialized (set to a starting value) as they’re declared:

```
char myFirstInitial = ‘j’;
int age = 30;
```

Some Built-In Types Of Variables In Java

Type	Description
byte	8 bit signed integer
short	16 but signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed real number
double	64 bit signed real number
char	16 bit Unicode character (ASCII and beyond)
boolean	1 bit true or false value
String	A sequence of characters between double quotes ("")

Location Of Variable Declarations

```
public class <name of class>
{
    public static void main (String[] args)
    {
            // Local variable declarations occur here

            << Program statements >>
                    :               :


    }
}
```

Style Hint: Initializing Variables

- Always initialize your variables prior to using them!
  - Do this whether it is syntactically required or not.
- Example how not to approach:

```
public class OutputExample1
{
   public static void main (String [] args)
   {
       int num;
       System.out.print(num);
    }
}
```

```
OutputExample1.java:7: error: variable num
might not have been initialized
      System.out.print(num);
                       ^
```

Java Constants

Reminder: constants are like variables in that they have a name and store a certain type of information but unlike variables they CANNOT change. (Unlike Python this is syntactically enforced…hurrah!).

Format:

final <*constant type*> <*CONSTANT NAME*> = <*value*>;

Example:

```
final int SIZE = 100;
```

Location Of Constant Declarations

```
public class <name of class>
{
    public static void main (String[] args)
    {
            // Local constant declarations occur here (more later)
            // Local variable declarations

            < Program statements >>
                        :          :


    }
}
```

Why Use Constants?

1. They make your program easier to read and understand

```
populationChange = (0.1758 – 0.1257) *
    currentPopulation;
```

Vs.

```
final float BIRTH_RATE = 17.58;
final float MORTALITY_RATE = 0.1257;
int currentPopulation = 1000000;
populationChange = (BIRTH_RATE -
    MORTALITY_RATE) * currentPopulation;
```

Why Use Constants? (2)

2. It can make your program easier to maintain (update with changes).
   - If the constant is referred to several times throughout the program, changing the value of the constant once will change it throughout the program.

Why Use Constants? (3)

```
final float BIRTH_RATE = 0.1758;
final float MORTALITY_RATE = 0.1257;
float populationChange = 0;
float currentPopulation = 1000000;
populationChange = (BIRTH_RATE - MORTALITY_RATE) * currentPopulation;
if (populationChange > 0)
    System.out.println("Increase“)
    System.out.println("Birth rate:“+ BIRTH_RATE + " Mortality rate:“ +
        MORTALITY_RATE, " + Population change:“ + populationChange);
else if  (populationChange < 0)
    System.out.println("Decrease“);
    System.out.println("Birth rate:“+BIRTH_RATE, “+Mortality rate:“+ MORTALITY_RATE
        +"Population change:“+populationChange);
else
    System.out.print("No change“);
    System.out.print("Birth rate:“+BIRTH_RATE, “+Mortality rate:“+ MORTALITY_RATE+
        "Population change:“+populationChange);
```

Why Use Constants? (4)

One change in the initialization of the constant changes all references to that constant.

```
final float BIRTH_RATE = 0.5;
final float MORTALITY_RATE = 0.1257;
float populationChange = 0;
float currentPopulation = 1000000;
populationChange = (BIRTH_RATE - MORTALITY_RATE) * currentPopulation;
if (populationChange > 0)
    System.out.println("Increase“)
    System.out.println("Birth rate:“+ BIRTH_RATE + " Mortality rate:“ +
        MORTALITY_RATE, " + Population change:“ + populationChange);
else if  (populationChange < 0)
    System.out.println("Decrease“);
    System.out.println("Birth rate:“+BIRTH_RATE, “+Mortality rate:“+ MORTALITY_RATE
        +"Population change:“+populationChange);
else
    System.out.print("No change“);
    System.out.print("Birth rate:“+BIRTH_RATE, “+Mortality rate:“+ MORTALITY_RATE+
        "Population change:“+populationChange);
```

Variable Naming Conventions In Java

- Compiler requirements
  - Can't be a keyword nor can the names of the special constants: true, false or null be used
  - Can be any combination of letters, numbers, underscore or dollar sign (first character must be a letter or underscore)

- Common stylistic conventions
  - The name should describe the purpose of the variable
  - Avoid using the dollar sign
  - With single word variable names, all characters are lower case
    - e.g., double grades;
  - Multiple words are separated by capitalizing the first letter of each word except for the first word
    - e.g., String firstName = "James";

Java Keywords

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

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
1	expression++ expression--	Post-increment Post-decrement	Right to left
2	++expression --expression + - ! ~ (type)	Pre-increment Pre-decrement Unary plus Unary minus Logical negation Bitwise complement Cast	Right to left

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
3	* / %	Multiplication Division Remainder/modulus	Left to right
4	+ -	Addition or String concatenation Subtraction	Left to right
5	<< >>	Left bitwise shift Right bitwise shift	Left to right

Common Java Operators / Operator Precedence

Precedence level	**Operator**	**Description**	**Associativity**
6	< <= > >=	Less than Less than, equal to Greater than Greater than, equal to	Left to right
7	= = !=	Equal to Not equal to	Left to right
8	&	Bitwise AND	Left to right
9	^	Bitwise exclusive OR	Left to right

Common Java Operators / Operator Precedence

Precedence level	**Operator**	**Description**	**Associativity**
10	|	Bitwise OR	Left to right
11	&&	Logical AND	Left to right
12	||	Logical OR	Left to right

Common Java Operators / Operator Precedence

Precedence level	**Operator**	**Description**	**Associativity**
13	=	Assignment	Right to left
	+=	Add, assignment	
	-=	Subtract, assignment	
	*=	Multiply, assignment	
	/=	Division, assignment	
	%=	Remainder, assignment	
	&=	Bitwise AND, assignment	
	^=	Bitwise XOR, assignment	
	|=	Bitwise OR, assignment	
	<<=	Left shift, assignment	
	>>=	Right shift, assignment	

Post/Pre Operators

The name of the online example is: Order1.java

```
public class Order1
{
    public static void main (String [] args)
    {
        int num = 5;
        System.out.println(num);
        num++;
        System.out.println(num);
        ++num;
        System.out.println(num);
        System.out.println(++num);
        System.out.println(num++);
    }
}
```

Post/Pre Operators (2)

The name of the online example is: Order2.java

```
public class Order2
{
    public static void main (String [] args)
    {
        int num1;
        int num2;
        num1 = 5;
        num2 = ++num1 * num1++;
        System.out.println("num1=" + num1);
        System.out.println("num2=" + num2);
    }
}
```

Unary Operator/Order/Associativity

The name of the online example: Unary_Order3.java

```
public class Unary_Order3.java
{
    public static void main (String [] args)
    {
        int num = 5;
        float fl;
        System.out.println(num);
        num = num * -num;
        System.out.println(num);
    }
}
```

Accessing Pre-Created Java Libraries

- It’s accomplished by placing an ‘import’ of the appropriate library at the top of your program.
- **Syntax:**

  import <*Full library name*>;

- **Example:**

```
import java.util.Scanner;
```

Getting Text Input

- You can use the pre-written methods (functions) in the Scanner class.

```
import java.util.Scanner;
```

- **General structure**:

```
main (String [] args)
{
    Scanner <name of scanner> = new Scanner (System.in);
    <variable> = <name of scanner> .<method> ();
}
```

Creating a scanner object (something that can scan user input)

Using the capability of the scanner object (actually getting user input)

Getting Text Input (2)

The name of the online example: MyInput.java

```
import java.util.Scanner;

public class MyInput
{
    public static void main (String [] args)
    {
        String str1;
        int num1;
        Scanner in = new Scanner (System.in);
        System.out.print ("Type in an integer: ");
        num1 = in.nextInt ();
        System.out.print ("Type in a line: ");
        in.nextLine ();
        str1 = in.nextLine ();
        System.out.println ("num1:" +num1 +"\t str1:" + str1);
    }
}
```

Useful Methods Of Class Scanner[1]

- nextInt ()
- nextLong ()
- nextFloat ()
- nextDouble ()
- nextLine ();

1 Online documentation: http://java.sun.com/javase/6/docs/api/

Reading A Single Character

- Text menu driven programs may require this capability.
- Example:

```
GAME OPTIONS
(a)dd a new player
(l)oad a saved game
(s)ave game
(q)uit game
```

- There’s different ways of handling this problem but one approach is to extract the first character from the string.
- Partial example:

```
String s = "boo“;
System.out.println(s.charAt(0));
```

Reading A Single Character

- Name of the (more complete example): MyInputChar.java

```
import java.util.Scanner;
public class MyInputChar
{
    public static void main (String [] args)
    {
        final int FIRST = 0;
        String selection;
        Scanner in = new Scanner (System.in);
        System.out.println("GAME OPTIONS");
        System.out.println("(a)dd a new player");
        System.out.println("(l)oad a saved game");
        System.out.println("(s)ave game");
        System.out.println("(q)uit game");
        System.out.print("Enter your selection: ");
```

Reading A Single Character (2)

```
        selection = in.nextLine ();
        System.out.println ("Selection: " + selection.charAt(FIRST));
    }
}
```

Decision Making In Java

- Java decision making constructs
    - if
    - if, else
    - if, else-if
    - switch

Decision Making: Logical Operators

Logical Operation	**Python**	**Java**
AND	and	&&
OR	or	||
NOT	not, !	!

Decision Making: If

Format:

```
if (Boolean Expression)
      Body
```

Example:

```
if (x != y)
      System.out.println("X and Y are not equal");

if ((x > 0) && (y > 0))
{
      System.out.println("X and Y are positive");
}
```

- Indenting the body of the branch is an important stylistic requirement of Java but unlike Python it is not enforced by the syntax of the language.
- What distinguishes the body is either:
    1. A semi colon (single statement branch)
    2. Braces (a body that consists of multiple statements)

Decision Making: If, Else

Format:

```
if (*Boolean expression*)
    *Body of if*
else
   *Body of else*
```

Example:

```
if (x < 0)
     System.out.println("X is negative");
else
    System.out.println("X is non-negative");
```

Example Program: If-Else

- Name of the online example: BranchingExample1.java

```
import java.util.Scanner;

public class BranchingExample1
{
   public static void main (String [] args)
   {
      Scanner in = new Scanner(System.in);
      final int WINNING_NUMBER = 131313;
      int playerNumber = -1;

      System.out.print("Enter ticket number: ");
      playerNumber = in.nextInt();
      if (playerNumber == WINNING_NUMBER)
        System.out.println("You're a winner!");
      else
        System.out.println("Try again.");
   }
}
```

If, Else-If

Format:

```
if (Boolean expression)
    Body of if
else if (Boolean expression)
  Body of first else-if
      :      :      :
else if (Boolean expression)
  Body of last else-if
else
   Body of else
```

If, Else-If (2)

Name of the online example: BranchingExample.java

```
import java.util.Scanner;

public class BranchingExample2
{
   public static void main (String [] args)
   {
      Scanner in = new Scanner(System.in);
      int gpa = -1;
      System.out.print("Enter letter grade: ");
      gpa = in.nextInt();
```

If, Else-If (3)

```
        if (gpa == 4)
           System.out.println("A");
        else if (gpa == 3)
           System.out.println("B");
        else if (gpa == 2)
           System.out.println("C");
        else if (gpa == 1)
           System.out.println("D");
        else if (gpa == 0)
           System.out.println("F");
        else
           System.out.println("Invalid letter grade");
     }
  }
```

Branching: Common Mistakes

- Recall that for single bodies: what lies between the closing bracket of the Boolean expression and the next semi-colon is the body.

body

```
if (Boolean Expression)
   instruction;
```

body

```
if (Boolean Expression) instruction;
```

```
if (Boolean Expression)
instruction1;
Instruction2;
```

body

Branching: Now What Happens???

```
if (Boolean Expression):
   instruction1;
instruction2;
```

Alternative To Multiple Else-If's: Switch

Format (character-based switch):

```
switch (character variable name)
{
     case '<character value>':
          Body
          break;

     case '<character value>':
          Body
          break;
             :
    default:
         Body
}
```

Important! The break is mandatory to separate Boolean expressions (must be used in all but the last)

1 The type of variable in the brackets can be a byte, char, short, int or long

Alternative To Multiple Else-If’s: Switch (2)

Format (integer based switch):

```
switch (integer variable name)
{
      case <integer value>:
           Body
           break;

      case <integer value>:
           Body
           break;
              :
     default:
          Body
}
```

1 The type of variable in the brackets can be a byte, char, short, int or long

Switch: When To Use/When Not To Use

- Benefit (when to use):
  - It may produce simpler code than using an if-elseif (e.g., if there are multiple compound conditions)

Switch: When To Use/When Not To Use (2)

- Name of the online example: SwitchExample.java

```
import java.util.Scanner;

public class SwitchExample
{
    public static void main (String [] args)
    {
        final int FIRST = 0;
        String line;
        char letter;
        int gpa;
        Scanner in = new Scanner (System.in);
        System.out.print("Enter letter grade: ");
```

Switch: When To Use/When Not To Use (3)

```
line = in.nextLine ();
letter = line.charAt(FIRST);
switch (letter)
{
   case 'A':
   case 'a':
      gpa = 4;
      break;

   case 'B':
   case 'b':
      gpa = 3;
      break;

   case 'C':
   case 'c':
      gpa = 2;
      break;
```

Switch: When To Use/When Not To Use (4)

```
            case 'D':
            case 'd':
               gpa = 1;
               break;

            case 'F':
            case 'f':
               gpa = 0;
               break;

            default:
               gpa = -1;

         }
         System.out.println("Letter grade: " + letter);
         System.out.println("Grade point: " + gpa);
    }
}
```

Switch: When To Use/When Not To Use (5)

- When a switch can’t be used:
  - For data types other than characters or integers
  - Boolean expressions that aren’t mutually exclusive:
    - As shown a switch can replace an ‘if-elseif’ construct
    - A switch cannot replace a series of ‘if’ branches).
  - Example when not to use a switch:

```
if (x > 0)
    System.out.print(“X coordinate right of the origin”);
If (y  > 0)
    System.out.print(“Y coordinate above the origin”);
```

  - Example of when not to use a switch:

```
String name = in.readLine()
switch (name)
{

}
```

Switch Example: Modified

- What happens if all the ‘break’ instructions have been removed?

Loops

Java Pre-test loops

- For
- While

Java Post-test loop

- Do-while

While Loops

Format:

```
while (Boolean expression)
   Body
```

Example:

```
int i = 1;
while (i <= 4)
{
     // Call function
     createNewPlayer();
     i = i + 1;
}
```

For Loops

Format:

```
for (initialization; Boolean expression; update control)
    Body
```

Example:

```
for (i = 1; i <= 4; i++)
{
    // Call function
    createNewPlayer();
    i = i + 1;
}
```

Post-Test Loop: Do-While

- Recall: Post-test loops evaluate the Boolean expression after the body of the loop has executed.
- This means that post test loops will execute one or more times.
- Pre-test loops generally execute zero or more times.

Do-While Loops

Format:

```
do
    Body
while (Boolean expression);
```

Example:

```
char ch = 'A';
do
{
    System.out.println(ch);
    ch++;
}
while (ch <= 'K');
```

Contrasting Pre Vs. Post Test Loops

- Although slightly more work to implement the while loop is the most powerful type of loop.
- Program capabilities that are implemented with either a ‘for’ or ‘do-while’ loop can be implemented with a while loop.
- Implementing a post test loop requires that the loop control be primed correctly (set to a value such that the Boolean expression will evaluate to true the first it’s checked).

Example: Post-Test Implementation

- Name of the online example: PostTestExample.java

```
public class PostTestExample
{
  public static void main (String [] args)
  {
    final int FIRST = 0;
    Scanner in = new Scanner(System.in);
    char answer;
    String temp;
    do
    {
      System.out.println("JT's note: Pretend that we play our game");
      System.out.print("Play again? Enter 'q' to quit: ");
      temp = in.nextLine();
      answer = temp.charAt(FIRST);
    } while ((answer != 'q') && (answer != 'Q'));
  }
}
```

Example: Pre-Test Implementation

```
public class PreTestExample
{
   public static void main (String [] args)
   {
      final int FIRST = 0;
      Scanner in = new Scanner(System.in);
      char answer = ' ';
      String temp;
      while ((answer != 'q') && (answer != 'Q'))
      {
         System.out.println("JT's note: Pretend that we play our game");
         System.out.print("Play again? Enter 'q' to quit: ");
         temp = in.nextLine();
         answer = temp.charAt(FIRST);
      }
   }
}
```

Now What Happens???

```
import java.util.Scanner;

public class PreTestExample
{
   public static void main (String [] args)
   {
      final int FIRST = 0;
      Scanner in = new Scanner(System.in);
      char answer = ' ';
      String temp;
      while ((answer != 'q') && (answer != 'Q'))
         System.out.println("JT's note: Pretend that we play our game");
         System.out.print("Play again? Enter 'q' to quit: ");
         temp = in.nextLine();
         answer = temp.charAt(FIRST);
   }
}
```

After This Section You Should Now Know

- How Java was developed and the impact of it's roots on the language
- The basic structure required in creating a simple Java program as well as how to compile and run programs
- How to document a Java program
- How to perform text based input and output in Java
- The declaration of constants and variables
- What are the common Java operators and how they work
- The structure and syntax of decision making and looping constructs