1. Basic Programming Concepts
1. Basic Programming Concepts

- Why programming?
- Program development
- Built-in data types
- Type conversion
A human being should be able to
change a diaper,
plan an invasion,
butcher a hog,
conn a ship,
design a building,
write a sonnet,
balance accounts,
build a wall,
set a bone,
comfort the dying,
take orders,
give orders,
cooperate,
act alone,
solve equations,
analyze a new problem,
pitch manure,
program a computer,
cook a tasty meal,
fight efficiently, and
die gallantly.

Specialization is for insects.  
Robert A. Heinlein
Time Enough for Love (1973)
You need to know how to program

in order to be able to tell a computer what you want it to do.

**Naive ideal:** Natural language instructions.

```
"Please simulate the motion of N heavenly bodies, subject to Newton’s laws of motion and gravity."
```

**Prepackaged solutions (apps)** are great when what they do is what you want.

**Programming** enables you to make a computer do anything you want.

---

*First programmer*  
Ada Lovelace

*First computer*  
Analytical Engine

← well, *almost* anything (stay tuned)
Programming: telling a computer what to do

Programming

- Is not just for experts.
- Is a natural, satisfying and creative experience.
- Enables accomplishments not otherwise possible.
- The path to a new world of intellectual endeavor.

Challenges

- Need to learn what computers can do.
- Need to learn a programming language.

“Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”

– Don Knuth
Telling a computer what to do

Machine language
- Easy for computer.
- Error-prone for human.

Natural language
- Easy for human.
- Error-prone for computer.

High-level language
- Some difficulty for both.
- An acceptable tradeoff.

Adding two numbers (see TOY lecture)

```
for (int t = 0; t < 2000; t++)
{
    a[0] = a[11] ^ a[9];
    System.out.print(a[0]);
    for (int i = 11; i > 0; i--)
        a[i] = a[i-1];
}
```

Simulating an LFSR (see Prologue lecture)

But which high-level language?

Naive ideal: A single programming language for all purposes.
Our Choice: Java

Java features
- Widely used.
- Widely available.
- Continuously under development since early 1990s.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

Java economy
- Mars rover.
- Cell phones.
- Blu-ray Disc.
- Web servers.
- Medical devices.
- Supercomputing.
- ...
Our Choice: Java

Java features
• Widely used.
• Widely available.
• Continuously under development since early 1990s.
• Embraces full set of modern abstractions.
• Variety of automatic checks for mistakes in programs.

Facts of life
• No language is perfect.
• You need to start with some language.

“Our are only two kinds of programming languages: those people always [gripe] about and those nobody uses.”
— Bjarne Stroustrup

Our approach
• Use a minimal subset of Java.
• Develop general programming skills that are applicable to many languages.

It’s not about the language!
A rich subset of the Java language vocabulary

### built-in types
- int
- long
- double
- char
- String
- boolean

### operations on numeric types
- +
- -
- *
- /
- %
- ++
- --

### String operations
- +
- ""
- length()
- charAt()
- compareTo()
- matches()

### assignment
- =

### flow control
- if
- else
- for
- while

### object oriented
- static
- class
- public
- private
- new
- final
- tostring()
- main()

### Math methods
- Math.sin()
- Math.cos()
- Math.log()
- Math.exp()
- Math.pow()
- Math.sqrt()
- Math.min()
- Math.max()
- Math.abs()
- Math.PI

### System methods
- System.print()
- System.println()
- System.printf()

### our Std methods
- StdIn.read*()
- StdOut.print*()
- StdDraw.*()
- StdAudio.*()
- StdRandom.*()

### punctuation
- {
- }
- (  
- )
- ,
- ;
- <
- <=
- >
- >=
- ==
- !=

### comparisons
- true
- false
- !
- &&
- ||

### boolean operations
- a[]
- length
- new

### arrays
- type conversion methods
- Integer.parseInt()
- Double.parseDouble()
Anatomy of your first program

```java
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

text file named `HelloWorld.java`

program name

main() method

body of main()
(a single statement)
Anatomy of your next several programs

```java
public class MyProgram {
    public static void main(String[] args) {
        ...
    }
}
```

- **program name**: `MyProgram`
- **main() method**: `public static void main(String[] args) {
    ...
} }

(text file named MyProgram.java)

(body of main() (a sequence of statements))
Pop quiz on "your first program"

**Q.** Use common sense to cope with the following error messages.

```text
% javac MyProgram.java
% java MyProgram
Main method not public.

% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
   public static main(String[] args)
   ^
```
Pop quiz on "your first program"

Q. Use common sense to cope with the following error messages.

A. Must have forgotten “public”.

```
% javac MyProgram.java
% java MyProgram
Main method not public.
```

A. Check HelloWorld. Aha! Forgot “void”.

```
% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
  public static main(String[] args)
   ^
```

```java
public static void main(String[] args)
```
Three versions of the same program.

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Lesson: Fonts, color, comments, and extra space are not relevant in Java language.
Note on program style

Different styles are appropriate in different contexts.
• Integrated development environment
• Booksite
• Book
• Your code

Enforcing consistent style can
• Stifle creativity.
• Confuse style with language.

Emphasizing consistent style can
• Make it easier to spot errors.
• Make it easier for others to read and use code.
• Enable development environment to provide visual cues.

Bottom line for you: Listen to the person assigning your grade.

or your boss!

Program 1.1.1

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.print("Hello, World");
        System.out.println();
    }
}
```

This code is a Java program that accomplishes a simple task. It is traditionally a beginner’s first program. The box below shows what happens when you compile and execute the program. The terminal application gives a command prompt (% in this book) and executes the commands that you type (% and then java in the example below). The result in this case is that the program prints a message in the terminal window (the third line).

Program 1.1.1 is an example of a complete Java program. Its name is HelloWorld.java, which means that the code resides in a file named HelloWorld.java (by convention in Java). The program’s sole action is to print a message back to the terminal window.

Note on program style

Enforcing consistent style can
• Stifle creativity.
• Confuse style with language.

Emphasizing consistent style can
• Make it easier to spot errors.
• Make it easier for others to read and use code.
• Enable development environment to provide visual cues.

Bottom line for you: Listen to the person assigning your grade.

or your boss!
**Image sources**

- http://commons.wikimedia.org/wiki/File:KnuthAtOpenContentAlliance.jpg
1. Basic Programming Concepts

- Why programming?
- Program development
- Built-in data types
- Type conversion
Program development in Java is a three-step process, *with feedback*

1. **EDIT your program**
   - Create it by typing on your computer's keyboard.
   - Result: a text file such as `HelloWorld.java`.

2. **COMPILE it to create an executable file**
   - Use the Java compiler
   - Result: a Java bytecode file such as `HelloWorld.class`
   - Mistake? Go back to 1. to fix and recompile.

3. **RUN your program**
   - Use the Java runtime.
   - Result: your program's output.
   - Mistake? Go back to 1. to fix, recompile, and run.

Diagram:

- **EDIT** to **COMPILE** to **RUN**
- A legal Java program:
  - ![Diagram](image)
- A legal Java program that does the wrong thing:
  - ![Diagram](image)
- Not a legal Java program:
  - ![Diagram](image)
Any creative process involves cyclic refinement/development.

A significant difference with programs: *We can use our computers to facilitate the process.*

**Program development environment:** Software for editing, compiling and running programs.

**Two time-tested options:** (Stay tuned for details).

**Virtual terminals**
- Same for many languages and systems.
- Effective even for beginners.
Bottom line: Extremely simple and concise.

**Integrated development environment**
- Often language- or system-specific.
- Can be helpful to beginners.
Bottom line: Variety of useful tools.
Program development environments: a very short history

**Historical context** is important in computer science.
- We regularly use old software.
- We regularly emulate old hardware.
- We depend upon old concepts and designs.

**Widely-used methods for program development**
- switches and lights
- punched cards/compiler/runtime
- editor/compiler/runtime/terminal
- editor/compiler/runtime/virtual terminal
- integrated development environment
Program development with switches and lights

Circa 1970: Use switches to input binary program code and data, lights to read output.

Stay tuned for details [lectures on the "TOY machine"]: 
Program development with punched cards and line printers

Mid 1970s: Use punched cards to input program code and data, line printer for output.

Ask your parents about the "computer center" for details.
Program development with timesharing terminals

Late 1970s: Use terminal for editing program, reading output, and controlling computer.

Timesharing allowed many users to share the same computer.
1980s to present day: Use multiple *virtual terminals* to interact with computer.

- **Edit** your program using any text editor in a virtual terminal.
- **Compile** it by typing `javac HelloWorld.java` in another virtual terminal.
- **Run** it by typing `java HelloWorld`
Program development with personal computers (another approach)

1980s to present day: Use a \textit{customized application} for program development tasks.

- \textbf{Edit} your program using the built-in text editor.
- \textbf{Compile} it by clicking the “compile” button.
- \textbf{Run} it by clicking the “run” button or using the pseudo-command line.

\textbf{“Integrated Development Environment” (IDE)}

http://drjava.org
## Software for program development: tradeoffs

<table>
<thead>
<tr>
<th>Virtual terminals</th>
<th>IDE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td><strong>Pros</strong></td>
</tr>
<tr>
<td>• Approach works with any language.</td>
<td>• Easy-to-use language-specific tools.</td>
</tr>
<tr>
<td>• Useful beyond programming.</td>
<td>• System-independent (in principle).</td>
</tr>
<tr>
<td>• Used by professionals.</td>
<td>• Used by professionals.</td>
</tr>
<tr>
<td>• Has withstood the test of time.</td>
<td>• Can be helpful to beginners.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Cons</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Good enough for long programs?</td>
<td>• Overkill for short programs?</td>
</tr>
<tr>
<td>• Dealing with independent applications.</td>
<td>• Big application to learn and maintain.</td>
</tr>
<tr>
<td>• Working at too low a level?</td>
<td>• Often language- or system-specific.</td>
</tr>
</tbody>
</table>

This course: Used in lectures/book.

Recommended for assignments.
Lessons from short history

Every computer has a program development environment that allows us to
• EDIT programs.
• COMPILE them to create an executable file.
• RUN them and examine the output.

Two approaches that have served for decades and are still effective:
• multiple virtual terminals.
Image sources

http://commons.wikimedia.org/wiki/Category:2013_Boston_Red_Sox_season#mediaviewer/
   File:Koji_Uehara_2_on_June_15,_2013.jpg
http://pixabay.com/p-15812/?no_redirect
1. Basic Programming Concepts