ICS 111
Review of the material until now

- Basic Programming Constructs
- Sequence
- Conditional
- Repetition/Loops
- Program Structure
- Parallel Execution

ICS 111
Basic Programming Constructs

- The basic steps of a computer are arithmetic and logical
- arithmetic operations: +, -, *, /, %
- arithmetic comparisons: ==, !=, <, <=, >, >=
- boolean operators: &&, ||, !
  - boolean expressions using && and || apply short-circuit evaluation, evaluating only the left operand if its value is sufficient to determine the result of the expression

ICS 111
Basic Programming: Strings

- Some of our basic operations work on strings
- `System.out.println, .print, .printf`
- given a String str, we can have:
- `int len = str.length();`
- `String sub = str.substring(`*int startIndex*`);`
- `String sub = str.substring(`*int startIndex, int nextIndex*`);`
- `char c = str.charAt(`*int index*`);`
- `boolean eq = str.equals(`*String s*`);`
- `int comparison = str.compareTo (`*String s*`);`
- each of these is a method in the String class
- calling a method is a primitive operation, just like the arithmetic and logical operations

ICS 111
Basic Programming: Math Library

- `java.lang.Math`
- familiar math functions, including powers, square root, and trigonometry
- Math.random()
- again, calling a method is a primitive operation, just like the arithmetic and logical operations

ICS 111
Variables and Types

- To remember the result of a computation (the value of an expression), we can use variables
- a variable declaration begins with a type followed by the variable name
- in ICS 111, every variable declaration must include the variable initialization

```
int x = 3;
String str = “hello world”;
boolean isGood = true;
```

ICS 111
Basic Programming: methods

- As well as using predefined methods, you can write your own methods
- `public static` *returnType methodName (parameters)*
- return type is void if the method doesn't return anything
- if the return type is not void, the method body must end with a return statement
- the method may contain arbitrary code
- the parameter list is a comma-separated list of variable declarations, with each variable initialized by the caller

ICS 111
Calling Methods

- the parameter list is a comma-separated list of variable declarations, with each variable initialized by the caller

  ```
  public String substring (int startIndex) { ...
  ```

- in the body of `substring`, `startIndex` can be treated like any variable
- when another piece of code calls (invokes)

  ```
  String sub = hello.substring(10)
  ```

  this is equivalent to initializing `startIndex = 10` at the beginning of the body of the method `substring`
- `sub` is initialized to whatever value `substring` returns

ICS 111
Some Special Words

- An **expression** (e.g. x + 2) defines a computation
- every expression has a **type** (e.g. int) and evaluates to a **value** (e.g. 10)
- A **variable** has a type and stores a value
- A variable must be **declared** before it can be used
- An **assignment** (eg. x = 3) stores a value into a variable
- An **initialization** is the first assignment to a variable
  - initialization is often (in ICS 111, always) done when the variable is declared
- Examples of variable declarations and initializations:
  - int x = 3;
  - String hello = “hello world”;
- A **method definition**, or **method declaration**, starts with the **method header**, including the return type, method name, and **parameters/arguments**, and continues with the **method body**
- A **statement** is the basic unit of execution. Statements we've seen so far include method calls, assignments, and compound statements including conditionals and loops

ICS 111
Sequence of Statements

- A sequence of statements is so natural that most programming languages define a sequence simply by writing the statements in the order to be executed
- The body of a method, of a conditional, or of a loop, is usually a sequence of declarations and statements

```
x = 3;
y = x + 7;
```

ICS 111
Conditionals

- Statements (and sequences of statements) can be executed conditionally, that is, only if a boolean expression is true

```
if (condition) { body of if }
else if (condition2) { body of else if }
else { body of else }
```

- the else if and the else parts are each optional

ICS 111
Switch Statements

- Test for several possible constant values at once

```
switch (x) {
case 1: ... break;
case 2: ... break;
case 3: ... break;
default: ... break;
}
```

- x is evaluated only once, which is convenient if x stands for a complicated expression such as Math.round(2 * Math.PI)
- remember to have break at the end of each case!
  - unless you want to “fall through”
- the switch/case notation is easier to read at a glance than a complicated set of if/else if/else statements
  - such as

```
if (x == 1) { ... } else if (x == 2) { ... } else if (x == 3) { ... } else {... }
```

ICS 111
Conditional Expressions

- If statements and switch statements provide conditional evaluation of statements
- What if you wanted to conditionally evaluate expressions?

  (condition) ? true-value : false-value

  ```
  char c = (x >= s.length()) ? ' ' : s.charAt(x);
  ```

- this either evaluates to the constant blank character ' ', or to the value of the expression `s.charAt(x)`

ICS 111
Loops

- Do “the same thing” multiple times
- “the same thing” will actually do something different each time, for example operate on a different user input, or at a different index of a string
- Loop while a condition is true
  - stop when the condition becomes false

ICS 111
While Loops

```
while (condition) { body of loop }
```

- to prevent infinite loops, the body of the loop must affect the condition

ICS 111
Trivia: replacing if with while

- `if (`*condition*`) {` *body of if* `}`
- can be rewritten as

```
while (condition) {
   body of if;
   condition = false;
}
```

- This is less clear than using if, so is never used in practice

ICS 111
Do...While Loops

```
do {
  body of loop
} while (condition);
```

- the body of the loop is executed at least once

ICS 111
For Loops

```
for (initialization; condition; update) {
  body of loop
}
```

- the initialization is done before the beginning of the loop, and may declare variables local to the loop
- the condition is tested before every execution of the loop
- the update is performed at the end of each loop

ICS 111
`break` and `return`

- break and return statements are similar in ending execution of the enclosing loop or switch statements (break) or the enclosing method (return)
- return statements must specify a value if the method return type is not `void`

ICS 111
Syntax and Semantics

- Programming languages have syntax and semantics
- Syntax defines what programs are legal
- Semantics defines the meaning of legal programs
- A syntactically correct program may compile and execute, but may not give the desired result

ICS 111
Program Structure

- In writing correct programs, it is helpful to make programs as clear as possible
- Comments and meaningful variable and method names help explain the writer's thinking
- Creating methods to handle sub-tasks of a complicated program makes it easier to understand both the main part of the program and the methods
  - it should be possible to understand each method in isolation
  - the main part of the program calls these methods

ICS 111
Program Structure: Methods

- If a program can be divided into clear, understandable parts, each part can be implemented as a separate method
- Then the main part of the program simply calls the individual methods
- Methods that are called from multiple places in the code provide **code reuse**
- Code reuse is advantageous because the work of writing and debugging a single method can be used more than once

ICS 111
Program Structure: Understanding

- You **must** understand how a program will compute its results before you can write the program
  - Clear thinking is the key
  - The book has several examples relating thinking and coding, including:
    - first do it by hand
    - tracing programs
    - test cases
    - flowcharts
    - storyboards
- When you are first learning to program, you are learning the thinking as well as the programming
- Errors and mistakes are a good sign that you are trying something you haven't done before
  - or that you are human...
- Compiler errors are helpful in fixing syntax errors, and some simple semantic errors

ICS 111
Program Structure: Visual

- Correct indentation helps in understanding programs
- Smaller methods are easier to understand than longer methods
- If a method or a variable has an understandable purpose, it should be easy to find a meaningful name for it

ICS 111
Parallel Execution

- Similar to a sequence, but statements may be executed in any order
- Best for statements that do not depend on each other:
  - `x = x + 1;` and `y = y + 2;` can be executed in parallel
  - but `x = x + 1;` and `boolean g = x > 0;` should be executed sequentially
- Parallel execution improves performance when your processor is multicore: multiple statements can be executed simultaneously
- This class doesn't focus on performance! Nor on parallel execution
- ICS 211 will start to consider performance

Summary

- Many basic building blocks, including all the method calls
- Building blocks are combined using sequences, loops, and conditionals to write useful programs
- Methods abstract away understandable functions which can be used without knowing the details of the implementation