

Revision

- To write programs that are easy to read, methods are very important.
- By making sure that programs call methods in their main part, we keep the main part as the overall structure of the program.
- We already know how to define a method, since `main` is actually a method. So basically we write `public static` followed by the return type, and then name and the formal argument list.
- We also know how to call a method, since we have been doing so for mathematical methods for a long time. Basically we write out the name of the method followed by a pair of parentheses enclosing the values to give the methods.

0911B L14-1

A simple method with return and how to use it

- A method can return something back to the caller by using the *return Type* and the `return` statement. E.g. a method to calculate the square of a double value:

```
public static double square(double num) {
    return num * num;
}
```

- A caller can get the value from the method, since the method call is an expression and has a value. For example:

```
public static void main(String[] args) {
    for (int i = 1; i <= 10; ++i) {
        System.out.println("The square of " + i + " is " + square(i) + ".");
    }
}
```

0911B L14-2

One more methods with return

If the method want to give a value back to the caller, it uses the `return` statement. The thing returned must be of type `returnType` in the method declaration.

```
/* Find all primes up to a limit */
import chapman.io.*;
public class FindPrimes {
    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("Largest number to test = ");
        int largest_num = in.readInt();
        for (int i=2; i<=largest_num; ++i)
            if (isPrime(i))
                System.out.println(i);
    }
    /* Return true if num is prime, and false otherwise. */
    public static boolean isPrime(int num) {
        for (int i = 2; i <= num-1; i++)
            if (num % i == 0)
                return false;
        return true;
    }
}
```

0911B L14-3

More on method calling: pass by value

- A method can have some formal arguments, where the actual arguments are given to the method when it is called.
- What happens if the arguments are modified? Will the variables of the caller be modified?
- The formal arguments work like local variable. So the variables of the caller won't be modified.
- We call this call-by-value. The formal arguments are **created** when the method is called, with the values **copied** from the formal arguments.

0911B L14-4

Some standard methods

The Java library contains lots of methods. Examples:

- `public static void System.exit(int return_code);`: exit the program immediately, giving a return code (usually 0).
- `public static boolean Character.isLetter(char ch);`: return true if `ch` is a letter, and false otherwise.
- Similar methods `isLetterOrDigit`, `isDigit`, `isSpaceChar`, `isUpperCase`, `isUpperCase`, etc.
- `public static char Character.toLowerCase(char ch);`: returns the lower case version of a character.
- Similar method `toUpperCase`.

0911B L14-5

Scope of variable: when they can be accessed

- Until now, all our variables are defined in a method.
- They are called "local" variables. A local variable is created (i.e. the memory is allocated, etc.) when their declaration statement is executed.
- The variable can only be used (accessed) within the method, and after they are declared. We say the scope of the variable is from the point of declaration until the end of the method.
- If you declare a variable in a `for`/compound statement, it can only be used within that `for`/compound statement respectively.
- Variable in the same method must have distinct names, except that two names in two independent `for`/compound statements may have the same name.

0911B L14-6

Class variables

- There is a type of variables, called class variables, that are not defined in a method.
- Class variables are defined inside a class but outside all methods.
- They have a keyword “static” in the declaration, like `static int word.size;`
- A class variables can be accessed by any method (unless restricted by other means). The method need not follow the declaration of the variable in order to use a class variable. (I.e., a function defined *before* the declaration of a class variable can still access the variable.)

0911B L14-7

Example use: Printing tree (again!!)

Using class variable for communication in printing tree:

```
import chapman.io.*;
public class Tree2 {
    // Left and right edge of '*'s
    static int left, right;

    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("Width of tree = ");
        int width = in.readInt();
        for (left = width, right = width;
            left >= 1;
            --left, ++right)
            printTreeLine();
    }

    // Print a tree line from left to right
    public static void printTreeLine() {
        for (int i = 1; i < left; i++)
            System.out.print(' ');
        for (int i = left; i <= right; i++)
            System.out.print('*');
        System.out.println();
    }
}
```

0911B L14-8

Duration of variable: when a variable will be destroyed?

- We have talked about when a variable is created: when the variable is declared for local variables, and when the program starts for class variables. So when are they destroyed?
- A local variable defined in a for/compound statement is destroyed at the end of the statement.
- A local variable not defined in a for/compound statement is destroyed at the end of the method.
- A class variable is destroyed only when the program finishes running.
- The “duration” of a variable refers to the time between it is created and it is destroyed. I.e., when the memory is still allocated to the variable.

0911B L14-9

Example on duration

What is the duration/scope of left? How about in? How about i?

```
import chapman.io.*;
public class Tree2 {
    // Left and right edge of '*'s
    static int left, right;

    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("Width of tree = ");
        int width = in.readInt();
        for (left = width, right = width;
            left >= 1;
            --left, ++right)
            printTreeLine();
    }

    // Print a tree line from left to right
    public static void printTreeLine() {
        for (int i = 1; i < left; i++)
            System.out.print(' ');
        for (int i = left; i <= right; i++)
            System.out.print('*');
        System.out.println();
    }
}
```

0911B L14-10

Class variable? Arguments and return values?

- Class variables and arguments/return values can both be used for methods to share values.
- Argument and return values explicitly specify that the caller sends a value to the method and receive a value back.
- Working on class variables is somewhat hidden: you don't know a method read or modify a class variable until you look at the code.
- Since we want to know that a value is passed when we actually read the program, we prefer arguments/return values.
- Use class variable only if the value is meaningful to the whole program. For each class variable, write comments to describe their meaning in the program.

0911B L14-11

Coding style

- For your program to be easily understood, your aim is not just to make the program compiles correctly.
- Good use of programming techniques makes your program “nicer”.
 - Follow conventions of variable, constant, method & class names.
 - Write comments for each class variable and methods.
 - Use more meaningful variable names for class variables.
 - Indentation your program correctly.
 - Use program constructs in ways that people expects.
 - Keep your methods short (e.g. within 30 lines).

0911B L14-12

Good code, bad code, horrible code

The same program to find all primes, but this time much more difficult to read:

```
import chapman.io.*;
public class FindPrimes {
    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("Largest number to test = ");
        int j = in.readInt();
        for (int i=2; i<=j; ++i) if (f(i)) System.out.println(i);
    }
    public static boolean f(int v) {
        for (int i = 2; i <= v-1; i++)
            if (v % i == 0) return false;
        return true;
    }
}
```

Try to write your programs so that it is easy to read, since you will be the one to read them most (when debugging your own programs)!

0911B L14-13

Little project: vending machine

Let's write a program for a soft-drink vending machine. But since we don't have a machine, we just print out that we drop coke, etc.

- The program runs forever. (Vending machines work like this!)
- It get coins, or get a button push for soft-drink, or a money-back lever.
- Once the amount is enough for the most expensive drink, it drops back all additional coins put into it.
- Once drink is chosen, it drop out the drink, and give back the changes—of course, in coins!

0911B L14-14

Class variables

- We can envision that we will have methods to get a coin, to make the changes, etc.
- All these methods need to know how much money has been put into the machine. So it makes sense to have a class variable here.
- Many will also want to read the input, so let's put the "in" variable as class variable as well.
- First few lines of the program:

```
/* Simulate a soft-drinks vending machine */
public class VendingMachine {
    static int amount_received = 0; // The total amount received, in cents
    static StdIn in = new StdIn(); // For getting input
    ...
}
```

0911B L14-15

Program skeleton

The program will repeatedly get some input, and then process the input. There are only three types of input: coin, button, money return. Thus the main program and getting input is simple.

```
public static void main(String[] args) {
    for (;;) {
        System.out.println("$" + amount_received/100.0 + " in machine.");
        char c = getInput();
        process(c);
    }
}

public static char getInput() {
    for (;;) {
        System.out.print("[C]oin insert, [B]utton press or [M]oney return? ");
        char ch = in.readChar();
        ch = Character.toLowerCase(ch);
        if (ch == 'c' || ch == 'b' || ch == 'm')
            return ch;
        System.out.println("Invalid input. Please try again.");
    }
}
```

0911B L14-16

Processing input

There are three types of inputs, each requiring quite difficult processing. So let's make all of them separate methods. The function process do just the following:

```
public static void process(char c) {
```

```
}
```

0911B L14-17

Inserting coin

When a coin is inserted, we check whether it is a "reasonable" coin, and then whether the amount is already a lot. In both case we reject it. If everything goes well, we modify amount_received.

```
public static void insertCoin() {
```

```
}
```

0911B L14-18

A button is pressed

When we get a button, we check whether the amount is enough. If so, we drop the drink and make the changes.

```
public static void buttonPress() {
```

```
}
```

0911B L14-19

Making changes

We adopt the following strategy in making changes: drop the coin of the greatest value smaller than amount_received, and repeat until it is 0.

```
public static void makeChanges() {
```

```
}
```

0911B L14-20