Previous Lecture:
- Class methods (static methods)
- Scope of variables
- OOP ideas, public vs private

Today’s Lecture:
- More about variables and their scope
- Method call and execution
- A simple loop

Assigned reading:
- T Sec 2.3, 2.4

OOP ideas
- Aggregate variables/methods into an abstraction (a class) that makes their relationship to one another explicit
- Objects (instances of a class) are self-governing (protect and manage themselves)
- Hide details from client, and restrict client’s use of the services
- Allow clients to create/get as many objects as they want

Variables and “default” values
- TWO main types of variables:
  - Primitive type
  - Reference to object

Some variables with different properties:
- **Local**: live and die inside a method
- **Instance variable**: owned by and accessed through individual instances (objects)
- **Static variable**: class variable shared by all instances—only one copy in a class

Variables and “default” values
- At declaration:
  - **Local**: No default value
  - **Instance variable**: has default value (the equivalent of zero)
  - **Static variable**: has default value (the equivalent of zero)

Scope of a variable
- The scope of a variable is the part of the program over which the variable is accessible
- A local variable is one that “lives and dies” in a method
- The scope of a local variable starts with its declaration and ends at the block in which it is declared
Which variables can `iMethod` access?
Starting from the nearest scope:

Which variables can `sMethod` access?
Starting from the nearest scope:

```java
/** Numeric interval */
class Interval {
    private double base;  // low end
    private double width; // interval width

    /** Getter method */
    public double getEnd() {
        return base + width;
    }

    /** Setter method */
    public void setWidth(int w) {
        width = w;
    }
}
```

Methods
A method is a named, parameterized group of statements

```
public double getEnd() {
    return base + width;
}
```

Value returned must have a type that is the same as the return type (or narrower)

Parameter list contains the type and name of each parameter.
I.e., a parameter is declared in the method—it is a variable local to the method
**return statement**

- Return-type `void` means nothing is returned from the method.
- There must be a `return` statement, unless return-type is `void`.
- You may use a `return` statement with return-type `void`:
  ```java
  return;
  ```
- A `return` statement may be used to end the execution of the method before the end of the method body is reached.

**Is returning the same as printing?**

- NO!
- Printing is like writing in the sand—you cannot keep what you’ve written.
- Returning means returning something to the program (so that that something can be used in the program).

```java
class Interval {
    private double base; // low end
    private double width; // interval width

    public double getEnd() {
        return base + width;
    }

    public void setWidth(int w) { width = w; }

    public static void main(String[] args) {
        Interval x = new Interval();
        x.setWidth(4);
        int rightSide = x.getEnd();
    }
}
```

### Areas of n-gon

**Inscribed hexagon**

\[
\text{Area} = \frac{n}{2} \sin \left(\frac{2\pi}{n}\right)
\]

**Circumscribed hexagon**

\[
\text{Area} = n \tan \left(\frac{\pi}{n}\right)
\]

/* Code segment to find and print areas of inscribed and circumscribed hexagons */

double innerA; // area of inscribed n-gon
double outerA; // area of circumscribed n-gon

innerA = (6/2.0) * Math.sin(2*Math.PI/6);
outerA = 6*Math.tan(Math.PI/6);

System.out.println("n\n\n| Inner Area | Outer Area |
---|---------|---------|
6 | innerA | outerA |
\nAreas of n-gons where n is 6...10

For each value of n, n is 6..10 calculate areas print areas
/* Code segment to find and print areas of
inscribed and circumscribed n-gons */
System.out.println(
    "n\tInner Area\tOuterArea\n");
double innerA; //area of inscribed n-gon
double outerA; //area of circumscribed n-gon
innerA= __/2.0)*Math.sin(2*Math.PI/__);
outerA= __*Math.tan(Math.PI/__); System.out.println(
    __ + innerA + \"\\t\" + outerA);

The for loop

Syntax:
for (initialization; condition; update)
    statement-to-repeat;

- Initialization, condition, and update are not
  required, but the semi-colons (;) are required

The for loop

Pattern for doing something \textit{n} times:

\begin{verbatim}
for ( int i=0; i<n; i++ ) {
    // do something
}
\end{verbatim}