Self-reference

- You can refer to any (instance) member of the current object within methods and constructors by using `this`.
- The most common reason for using the `this` keyword is because a field is shadowed by method parameters.
- You can also use `this` to call another constructor in the same class by invoking `this()`.
Example: Point (Revisited)

```java
class Point {
    ...
    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    ...
}
```

- Note that the `this` operator cannot be used in `static` methods.
Instance Members and Static Members

- You may notice that, until now, all members are declared w/o static.
- It means that each object has its own values with behaviors.
- The aforesaid members are called instance members.
- Note that these instance members are available only after the object is created.
Memory used by JVM

### Table: JVM Memory Areas

<table>
<thead>
<tr>
<th>Area</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Objects</td>
</tr>
<tr>
<td>Stack</td>
<td>Methods, Local Variables, Reference Variables</td>
</tr>
<tr>
<td>Code</td>
<td>Byte Code</td>
</tr>
<tr>
<td>Static</td>
<td>Static Methods and Data</td>
</tr>
</tbody>
</table>

- **-Xms**: The JVM min Heap Size
- **-Xmx**: The JVM max Heap Size
- **-Xss**: The Java Thread Stack Size, the default is OS and JVM dependent, and it can range 256k-to-1MB. The default should be tuned down to a range that doesn’t cause StackOverflow. I often use 128k-192k. Since the default -Xss is high, tuning it down can help save on memory used and given back to the Guest OS.

Perm Size is an area additional to the -Xmx (Max Heap) value and is not GC’ed because it contains class-level information.

“other mem” is additional mem required for NIO buffers, JIT code cache, classloaders, Socket Buffers (receive/send), JNI, GC internal info.

\[
\text{JVM Memory} = \text{JVM Max Heap} (-Xmx\text{ value}) + \text{JVM Perm Size} (-XX:MaxPermSize) + \text{NumberOfConcurrentThreads} \times (-Xss\text{ value}) + \text{“other mem”}
\]
Static Members

- The static members belong to the class\(^1\), and are shared between the instance objects.
- In other words, there is only one copy of the static members, no matter how many objects of the class are created.
- They are ready once the class is loaded.
- They can be invoked directly by the class name without using any instance.
- For example, Math.random().

\(^1\)Aka class members.

Zheng-Liang Lu
• A static method can access other static members. (Trivial.)
• However, static methods cannot access to instance members directly. (Why?)
• For example,

```java
... double getDistanceFrom(Point that) {
    return Math.sqrt(Math.pow(this.x - that.x, 2)
                        + Math.pow(this.y - that.y, 2));
}

static double distanceBetween(Point p1, Point p2) {
    // You cannot access to x and y directly!
    return Math.sqrt(Math.pow(p1.x - p2.x, 2)
                        + Math.pow(p1.y - p2.y, 2));
}
...```

Zheng-Liang Lu
Example: Count of Points

class Point {
    ...
    static int numOfPoints = 0;

    Point() {
        numOfPoints++;
    }

    Point(int x, int y) {
        this(); // calling the constructor with no input argument; should be placed in the first line in the constructor
        this.x = x;
        this.y = y;
    }
    ...
}
Exercise: Singleton

- In some situations, you may create the only instance of the class.

```java
class Singleton {
    // Will be ready as soon as the class is loaded.
    private static Singleton INSTANCE = new Singleton();

    // Do now allow to invoke the constructor by other classes.
    private Singleton() {}

    // Only way to obtain this singleton by the outside world.
    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

See any textbook for design patterns.
Garbage Collection (GC)³

• Java handles deallocation automatically.
• Automatic GC is the process of looking at the heap memory, identifying whether or not the objects are in use, and deleting the unreferenced objects.
• An object is said to be unreferenced if the object is no longer referenced by any part of your program.
  • Simply assign null to the reference to make the object unreferenced.
• So the memory used by these objects can be reclaimed.

³http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html
The method `finalize()` conducts a specific task that will be executed right before the object is reclaimed by GC.

The `finalize()` method can be only invoked prior to GC.

In practice, it must not rely on the `finalize()` method for normal operations. (Why?)
Example

```java
public class Garbage {
    private static int numOfObjKilled = 0;

    public void finalize() {
        numOfObjKilled++;
    }

    public static void main(String[] args) {
        double n = 1e7;
        for (int i = 1; i <= n; i++)
            new Garbage(); // lots of unreferenced objects
        System.out.println(numOfObjKilled);
    }
}
```

- You may try different number for instance creation.
- The number of the objects reclaimed by GC is uncertain.
HAS-A Relationship

- **Association** is a weak relationship where all objects have their own lifetime and there is no ownership.
  - For example, teacher ↔ student; doctor ↔ patient.
- If A uses B, then it is an **aggregation**, stating that B exists independently from A.
  - For example, knight ↔ sword; company ↔ employee.
- If A owns B, then it is a **composition**, meaning that B has no meaning or purpose in the system without A.
  - For example, house ↔ room.
Example: Lines

- +2: two **Point** objects used in one **Line** object.
More Examples

- Circle, Triangle, and Polygon.
- Book with Authors.
- Lecturer and Students in the classroom.
- Zoo with many creatures, say Dog, Cat, and Bird.
- Channels played on TV.
- More.
More Relationships Between Classes

- **Inheritance**: passing down states and behaviors from the parents to their children
- **Interfaces**: grouping the methods, which belongs to some classes, as an interface to the outside world
- **Packages**: grouping related types, providing access protection and name space management
First IS-A Relationship: Inheritance

- The relationships among Java classes form class hierarchy.
- We can define new classes by inheriting commonly used states and behaviors from predefined classes.
- A class is a subclass of some class, which is so-called the superclass, by using the extends keyword.
  - For example, B extends A.
- Semantically, we say B specializes A.
  - Equivalently, one subclass is a special case of the superclass.
  - For example, human and dog are two specific types of animals.
• Note that a class can extend only one other class, while each superclass has the potential for an unlimited number of subclasses.
• Alternatively, we say A generalizes B and C when both B and C are subclasses of A.
  • Code reuse again.
```java
class Animal {
    String name;
    int weight;

    Animal(String s, int w) { name = s; weight = w; }

    void eat() { weight++; }
    void exercise() { weight--; }
}

class Human extends Animal {
    Human(String s, int w) { super(s, w); }
    void writeCode() {}
}

class Dog extends Animal {
    Dog(String s, int w) { super(s, w); }
    void watchDoor() {}
}
```

See Fig. 3-1 in p. 113 of Evans and Flanagan.
• Recall that the keyword `this` is used to refer to the object itself.

• You can use the keyword `super` to refer to (non-private) members of the superclass.

• Note that `super()` can be used to invoke the constructor of its superclass, just similar to `this()`.
As the constructor is invoked, the constructor of its superclass is invoked accordingly.

You might think that there will be a whole chain of constructors called, all the way back to the constructor of the class Object, the topmost class in Java.

So every class is an immediate or a distant subclass of Object.

Recall that the method finalize() and toString() are inherited from Object.

- toString(): return a string which can be any information stored in the object.
```java
class A {
    A() { System.out.println("A is creating..."); }
}

class B extends A {
    B() { System.out.println("B is creating..."); }
    // overriding toString()
    public String toString() { return "I am B."; }
}

class ConstructorChainingDemo {
    public static void main(String[] args) {
        B b = new B();
        System.out.println(b);
    }
}
```

- The `println()` method (and similar methods) can take an object as input, and invoke `toString()` method implicitly.