Keyword this

- Can be used by any object to refer to itself in any class method
- Typically used to
  - Avoid variable name collisions
  - Pass the receiver as an argument
  - Chain constructors

Keyword this

- Keyword this allows a method to refer to the object that invoked it.
- It can be used inside any method to refer to the current object:

```
Box(double width, double height, double depth) {
this.width = width;
this.height = height;
this.depth = depth;
}
```

Garbage Collection

- Garbage collection is a mechanism to remove objects from memory when they are no longer needed.
- Garbage collection is carried out by the garbage collector:
- 1) The garbage collector keeps track of how many references an object has.
- 2) It removes an object from memory when it has no longer any references.
- 3) Thereafter, the memory occupied by the object can be allocated again.
- 4) The garbage collector invokes the finalize method.

finalize() Method

- A constructor helps to initialize an object just after it has been created.
- In contrast, the finalize method is invoked just before the object is destroyed:

1) implemented inside a class as:

```
protected void finalize() { … }
```

2) implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out

Constructor

- A constructor initializes the instance variables of an object.
- It is called immediately after the object is created but before the new operator completes.

  1) it is syntactically similar to a method:

  2) it has the same name as the name of its class

  3) it is written without return type
- When the class has no constructor, the default constructor automatically initializes all its instance variables with zero.

Example: Constructor

```
class Box {
double width;
double height;
double depth;
Box() {
System.out.println("Constructing Box");
width = 10; height = 10; depth = 10;
}
double volume() {
return (width * height * depth);
}
}
```

Parameterized Constructor

```
class Box {
double width;
double height;
double depth;
Box(double w, double h, double d) {
width = w; height = h; depth = d;
}
double volume()
{
    return width * height * depth;
}
}
```

Methods

- General form of a method definition:

```
type name(parameter-list) {
                                … return value;
                            …
                        }
```

- Components:

  1) type - type of values returned by the method. If a method does not return any value, its return type must be void.

  2) name is the name of the method

  3) parameter-list is a sequence of type-identifier lists separated by commas

  4) return value indicates what value is returned by the method.

Example: Method

- Classes declare methods to hide their internal data structures, as well as for their own internal use: Within a class, we can refer directly to its member variables:

```
class Box {
double width, height, depth;
void volume() {
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
}
```

Parameterized Method

- Parameters increase generality and applicability of a method:
- 1) method without parameters

```
int square()
    { return 10*10; }
```

- 2) method with parameters

```
int square(int i)
    { return i*i; }
```

- Parameter: a variable receiving value at the time the method is invoked.
- Argument: a value passed to the method when it is invoked.

Method Overloading

- It is legal for a class to have two or more methods with the same name.
- However, Java has to be able to uniquely associate the invocation of a method with its definition relying on the number and types of arguments.
- Therefore the same-named methods must be distinguished:

1) by the number of arguments, or

2) by the types of arguments

- Overloading and inheritance are two ways to implement polymorphism.

Example: Overloading

```
class OverloadDemo {
void test() {
System.out.println("No parameters");
}
void test(int a) {
System.out.println("a: " + a);
}
void test(int a, int b) {
System.out.println("a and b: " + a + " " + b);
}
double test(double a) {
System.out.println("double a: " + a);
return (a*a);
}
}
```

Constructor Overloading

```
class Box {
double width, height, depth;
Box(double w, double h, double d) {
width = w; height = h; depth = d;
}
Box() {
width = -1; height = -1; depth = -1;
}
Box(double len) {
width = height = depth = len;
}
double volume()
{ return width * height * depth; }
}
```