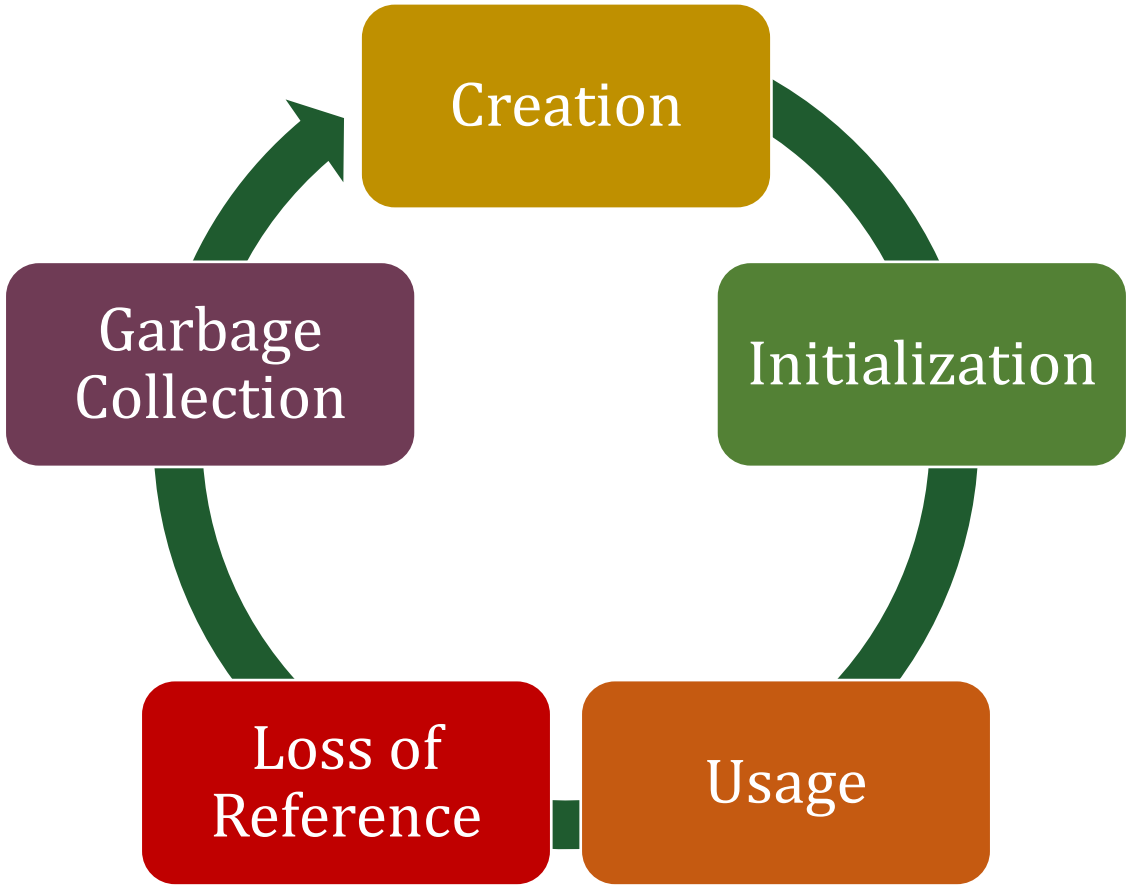
Object Life Cycle

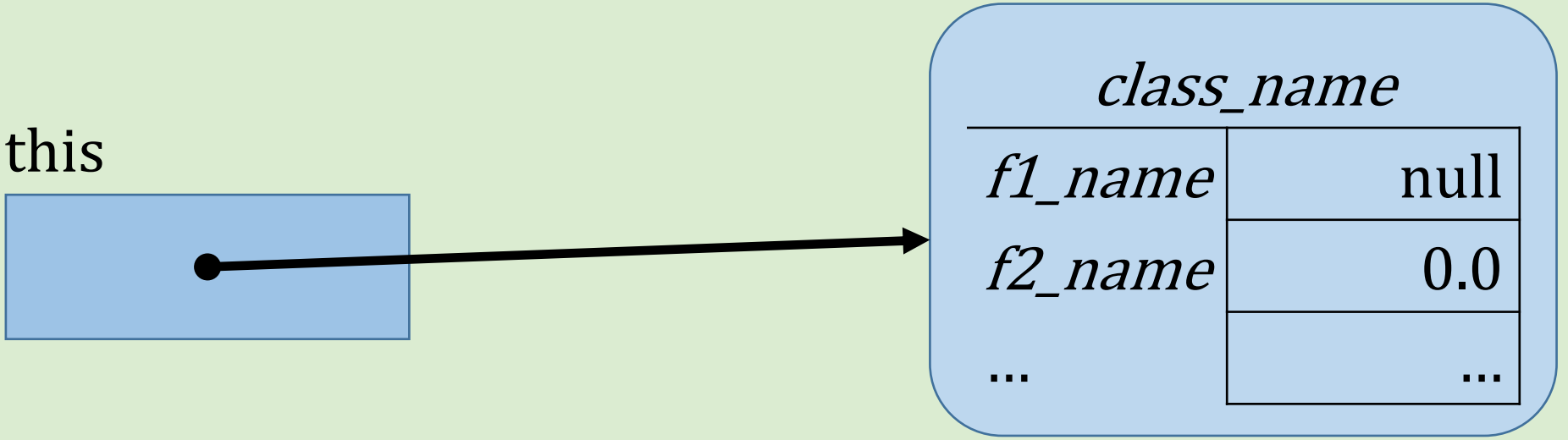


Defining Constructors

- A constructor is java code that is almost the same as a method
- The name of a constructor must be the class name itself!
- Since constructors are so much like methods, we will talk about constructors as if they are methods (but point out differences)

Constructor implicit "this" reference

- Constructors are like instance methods – never "static"
- When a constructor is invoked, the JVM has already created a new object in the correct class, using "heap" memory
- All the fields in that new object are initialized to zero or null
  - Numeric fields initialized to zero
  - Character fields initialized to null (unprintable empty character)
  - References are initialized to null (uninstantiated)



Constructor definition syntax

```
class classname {
    modifiers classname (parameters) {
        body
    }
}
```

No return type!

"method name" is class name

Cannot be "static"

No "return" statement.
Implicitly returns "this"

No-parameter Constructors are optional
If null/0 field initializations are OK.

Typical Constructor

```
class Rectangle {
        int x; int y; int width; int height;

        public Rectangle(int x,int y, int width, int height) {
                this.x=x; this.y=y;
                this.width=width; this.height=height;
        }
        …
}
```

Alternative constructor using “this”

Special Topic 3.1

```
class BankAccount {
        double value;
        public BankAccount(double value) {
                this.value = value;
                System.out.println("New account balance: " + value);
        }
        public BankAccount() {// Instead of the null constructor…
                this(0.0); // Invoke constructor w/ parameter
        }
```

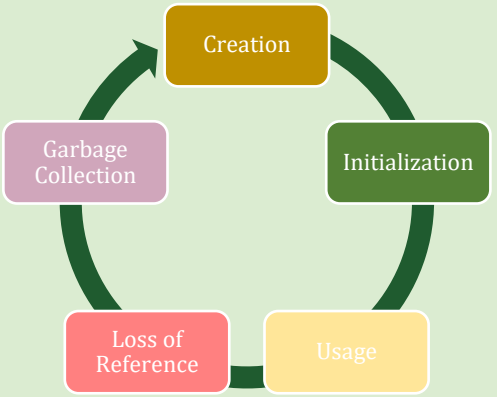
"this" refers to this class, not this object.

"this" **must** come first!

Example: java.awt.Rectangle

```
public Rectangle(int x, int y, int width, int height) {
        this.x=x; this.y=y; this.width=width; this.height=height; }
public Rectangle() { this(0,0,0,0); } // redundant null constructor
public Rectangle(Point p) { this(p.x,p.y,0,0); }
public Rectangle(int width, int height) { this(0,0,width,height); }
public Rectangle(Dimension d) { this(0,0,d.width,d.height); }
public Rectangle(Point p, Dimension d) {
        this(p.x,p.y,d.width,d.height); }
public Rectangle(Rectangle r) { this(r.x,r.y,r.width,r.height); }
```

Constructor Invocation

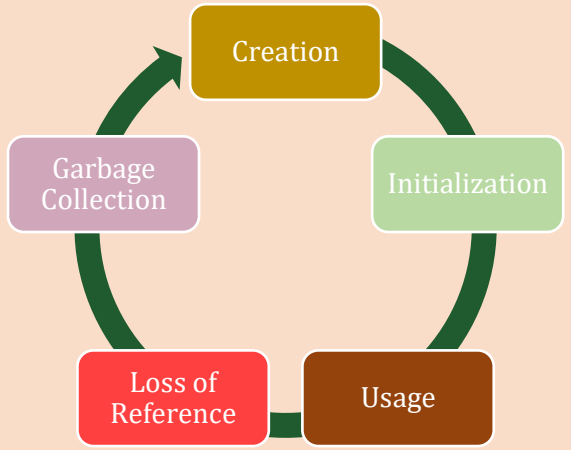


- Invoked as: new *class*(*constructor_arguments*);
  - *class* – Class of the newly created object
  - *constructor_arguments* – Any arguments required to create the object
- Constructor "method" name must match class name!
- Java creates a new object
  - All fields are initialized to zero or empty or null
- Constructor initializes field values
- Constructor implicitly returns a reference to this

Sect. 2.4

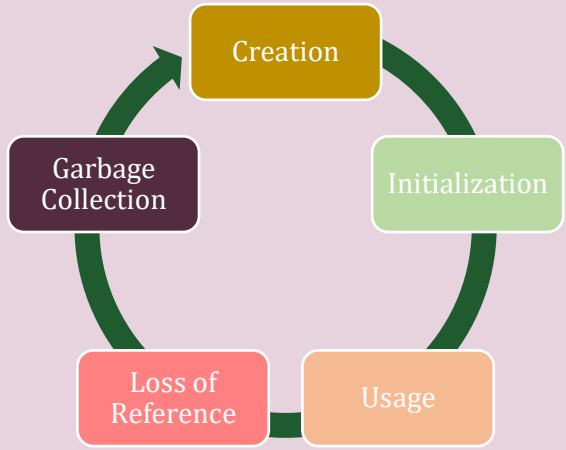
```
BankAccount checking = new BankAccount(249.37);
```

Using Objects



- You can use an object as long as you have a reference variable that references that object
- There may be several references to the same object
- References can be lost several ways
  - Reference variable can be re-assigned to a different object
  - Reference variables can go out of scope
  - Referencing object can lose all references
- Once an object becomes unreachable, e.g. no longer has a handle
  - No Java code can use that object anymore
  - The object becomes *available* for garbage collection

Garbage Collection



- Periodically, the JVM performs "garbage collection"
- Recycles all unreachable objects
  - Returns their memory so they can be used for other objects

- You don't need to delete objects
  - Just lose reference

- Simplifies coding
  - but causes run-time increase