Rounding Out Classes

The objectives of this chapter are:

- To discuss issues surrounding passing parameters to methods
- What is "this"?
- To introduce class variables and class methods
- To explain object deletion and the garbage collector

Passing Primitives as Parameters

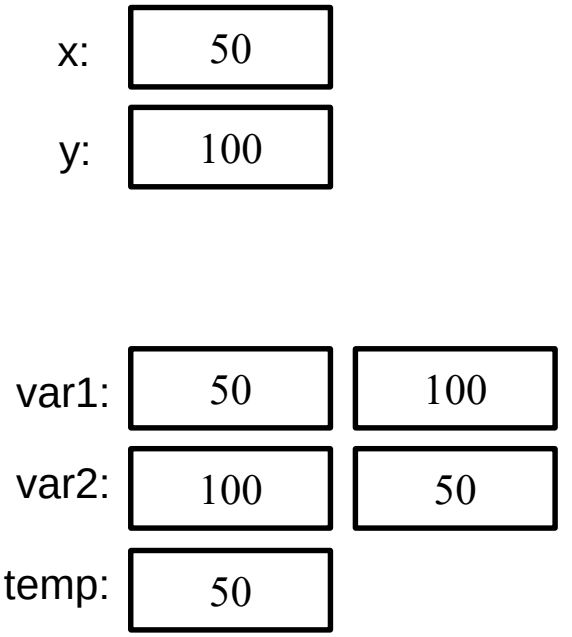
- Will the following code work?

```
public void method1()
{
  int x = 50;
  int y = 100;
  swap(x,y);
  System.out.println("x is:" + x + " y is:" + y);
}

public void swap(int var1, int var2)
{
  int temp = var1;
  var1 = var2;
  var2 = temp;
}
```

Passing Parameters by Value

- Parameters are passed by value in Java
  - The code does not behave as intended.

```
public void method1()
{
  int x = 50;
  int y = 100;
  swap(x,y);
}

public void swap(int var1, int var2)
{
  int temp = var1;
  var1 = var2;
  var2 = temp;
}
```

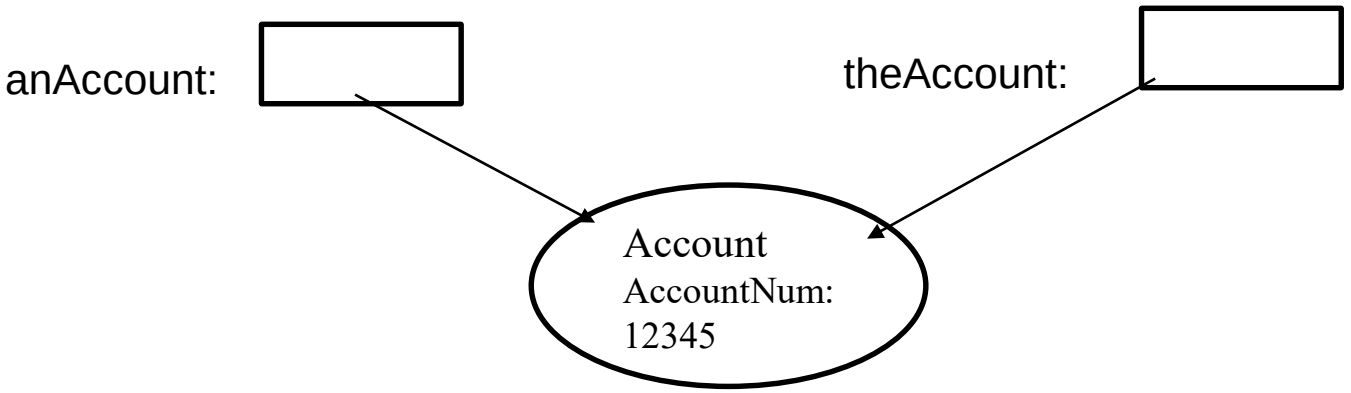


Passing Objects as Parameters

- It is also possible to pass object references as parameters to methods.
  - The reference itself is passed by value
  - The reference can be used to manipulate the target object
- Because of this behaviour, there are many classes that are defined as immutable (not changeable) or have instance variables which are immutable.
  - This allows the object's reference to be passed without violating encapsulation.
  - String is an example of an immutable class.
    - It does not have a set method for the String data.
    - String data can only be set upon initialization
  - Many business oriented classes have attributes which cannot be changed after initialization
    - Transaction
    - Account

Passing Objects as Parameters

```
public void method1()
{
  Account anAccount = new Account(12345);
  initializeAccount(anAccount)
}

public void initializeAccount(Account theAccount)
{
  Date openingDate = new Date();
  theAccount.setOpeningDate(openingDate);
  [... more initialization ...]
}
```



The this reference

- For every instance method, there is a "hidden" parameter called "this"
  - this is a reference to the instance (ie. object) upon which the method is being invoked
  - this can be used to access instance variables (although "this" is implied)
  - this can be **returned** from a method.

```
public class Employee
{
  private String name;

  public void setName(String aName)
  {
       this.name = aName;
       name = aName;                    // Equivalent
  }
}
```

Class Members

- Class Members:
  - Members declared in a class. Both data members and member functions.
- Class variables:
  - A data member which is declared with ***static*** modifier.
- Instance variables:
  - A data member in a class **without static** modifier
  - The main difference between the **class variable and Instance variable** is, first time when class is loaded in to memory, and then **only memory is allocated for all class variables**. That means, class variables not depends on the Objects of that class. Whatever numbers of objects are there, only one copy is created at the time of class loading. The main advantage of this class variable is, suppose if we want to maintain a counter to count the no of object of type that class, in that case we use variable counter as the static variable.
  - The compiler resolves the variable using the "this" reference.

Class Members Cont'd

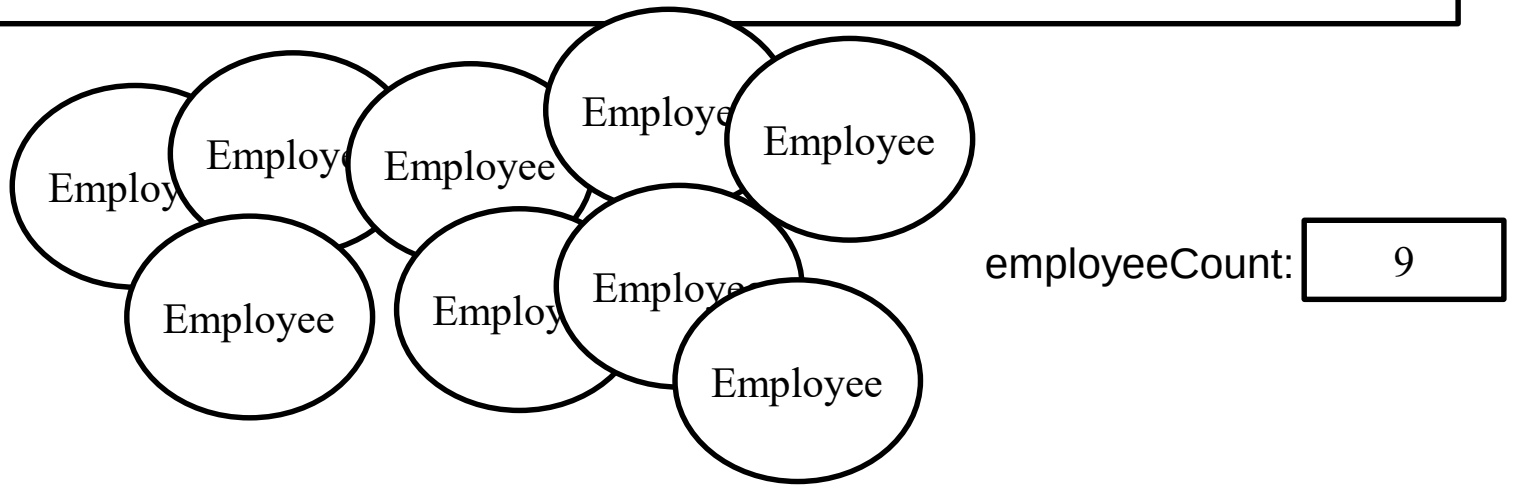
- Class method:
  - A method with ***static*** modifier.
- Instance methods:
  - A method without static modifier. For Both class methods and Instance method, memory allocated only once, at the time of loading the class. But difference is, we can access the static method, directly with class name, without using object of that class. Where as Instance method can be access through only object.

- Some restrictions with static methods:
  - ***Static*** method can use only ***static*** variables.
  - ***Static*** method cannot the non-static method
  - Each time an instance is created, instance variables are created as well.
- Class variables are NOT created each time an instance is created.
- Class variables can be seen as global variables available to all instances of the class.

Class Variables - Declaration

- Class variables are declared as "static" in the class definition

```
public class Employee
{
  private static int employeeCount = 0;
  private String name;

  public Employee()
  {
      employeeCount++;
  }

}
```



Class Methods

- Instance variables are accessed through Instance methods
- Class variables are accessed through Class methods
- Instance methods are invoked on instances (ie. Objects)
- Class methods are invoked on classes
  - Class methods do not have a "this" reference
- Class methods can only access Class variables
- Instance methods can access both Class variables and Instance variables.
- See the Java API Documentation for the Math class for examples.

The "main" Method

- You may recall that the "main" method is declared as static
- This means that it is a class method
  - It is invoked on a class, not an instance
  - When the JVM is started, there are no instances of any class. Therefore, "main" must be static
  - main does not have access to any instance variables.
  - Usually, the responsibility of the main method is to instantiate objects

```
public class HelloWorld
{
  private String message = "Hello World";

  public static void main(String[] args)
  {
      System.out.println(message);        // ERROR!
  }
}
```

Redundant Code.

- One of the design goals of O-O is to reduce redundant code
  - Method overloading actually encourages redundant code
  - Redundant constructors means that initialization code will appear in multiple locations

```
public class Account
{
  private String owner;
  private int accountNumber;

  public Account()
  {
        owner = "Unknown";
        accountNumber = 0;
  }
  public Account(String ownersName)
  {
        owner = ownersName;
        accountNumber = 0;
  }

  public Account(String ownersName, int anAccountNumber)
  {
        owner = ownersName;
        accountNumber = anAccountNumber;
  }
```

Using "this" to reduce code redundancy

- Constructors can invoke other constructors by using "this" as a method invocation
  - It must be the first line of code in the constructor

```
public class Account
{
  private String owner;
  private int accountNumber;

  public Account()
  {
       this("Unknown", 0);
  }
  public Account(String ownersName)
  {
       this(ownersName, 0);
  }

  public Account(String ownersName, int anAccountNumber)
  {
       owner = ownersName;
       accountNumber = anAccountNumber;
  }
```

Garbage Collection

- Each object maintains a reference count.
- In order to use an object, one must have a reference to that object.
- If an object has no references, it is no longer usable.
- The garbage collector will reclaim any memory resources being consumed by unreferenced objects.
- The user/programmer has no control over when the garbage collector runs. It normally runs when:
  - No other threads are running in the VM
  - The amount of free space available goes below a threshold value.

Easy Engineering Classes – Free YouTube Coaching

For Engineering Students of GGSIPU, UPTU and Other Universities, Colleges of India

Java Garbage Collection

↳ means unreferenced objects.

→ It is the process of reclaiming the runtime unused memory automatically.

Advantages

i) Makes Java Memory Efficient

ii) Automatically done. (JVM)

What is the meaning of Unrefereced object?

i) By nulling the reference →
```
A a = new A();
a = null;
```

ii) By assigning ref. -lo another →
```
A b = new A();
a = b;
```

iii) By anonymous object. →
```
new A();
```

object → Methods. ← System and Runtime.

- finalize() — before object is garbage collected.
- gc() — ↳ invoke garbage coll^{n}.

Example

```
class C1 { p.s.v.m(String[] ar){
    C1 a = new C1();
    C1 b = new C1();
    a = null; b = null;
    System.gc();
}
public void finalize()
{ S.O.P("G.C");
}
}
```

o/p:

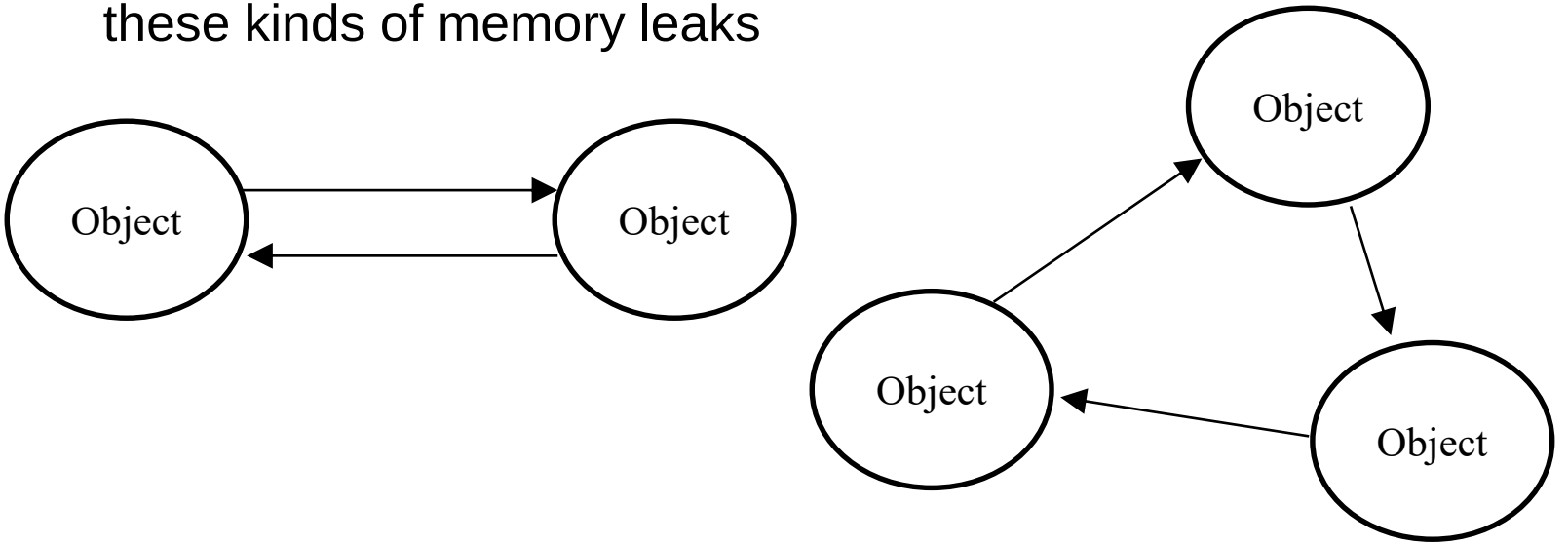
G.C

G.C

Are memory leaks possible in Java?

- Many people, erroneously, believe that memory leaks are not possible in Java.
  - They are possible, they are just more unlikely

- A memory leak can occur when two or more objects refer to each other, but there are no "external" references to any of those objects.
  - As a guideline, avoiding bidirectional associations will help to avoid these kinds of memory leaks



Garbage Collection - finalize()

- Since objects can hold scarce resources (ie. open file, connection to database, etc), it is important to release those resources when an object is destroyed.

- When the garbage collector is ready to return an object's memory to the system, it invokes the object's finalize() method.
  - Place any cleanup code in that method

- Remember, the programmer has no control over the garbage collector. As a result, the programmer has no control over when this method will be invoked (if ever).

Review

- Parameters are passed by value or by reference?
- What happens when an object is passed as a parameter to a method? What happens when an object is returned from a method?
- What is "this"?
- What are class variables and how are they defined?
- What are class methods?
- How does one constructor invoke another?
- What is garbage collection?
- What is the purpose of the finalize() method?