# Week 6: Topics

- `static` Keyword in Java

# `static` Keyword in Java

- The `static` keyword is used in Java mainly for memory management
  - Used with variable and methods

- It is used for a *constant* variable or a method

  That is the same for every instance of a class

- On the other hand, every object has its own copy of

  All the instance variable of a class

# A quick tutorial:

```
Blackboard: Week6/Counter_1
```

# `static` Variables in Java

- In certain cases, only one copy of a particular variable

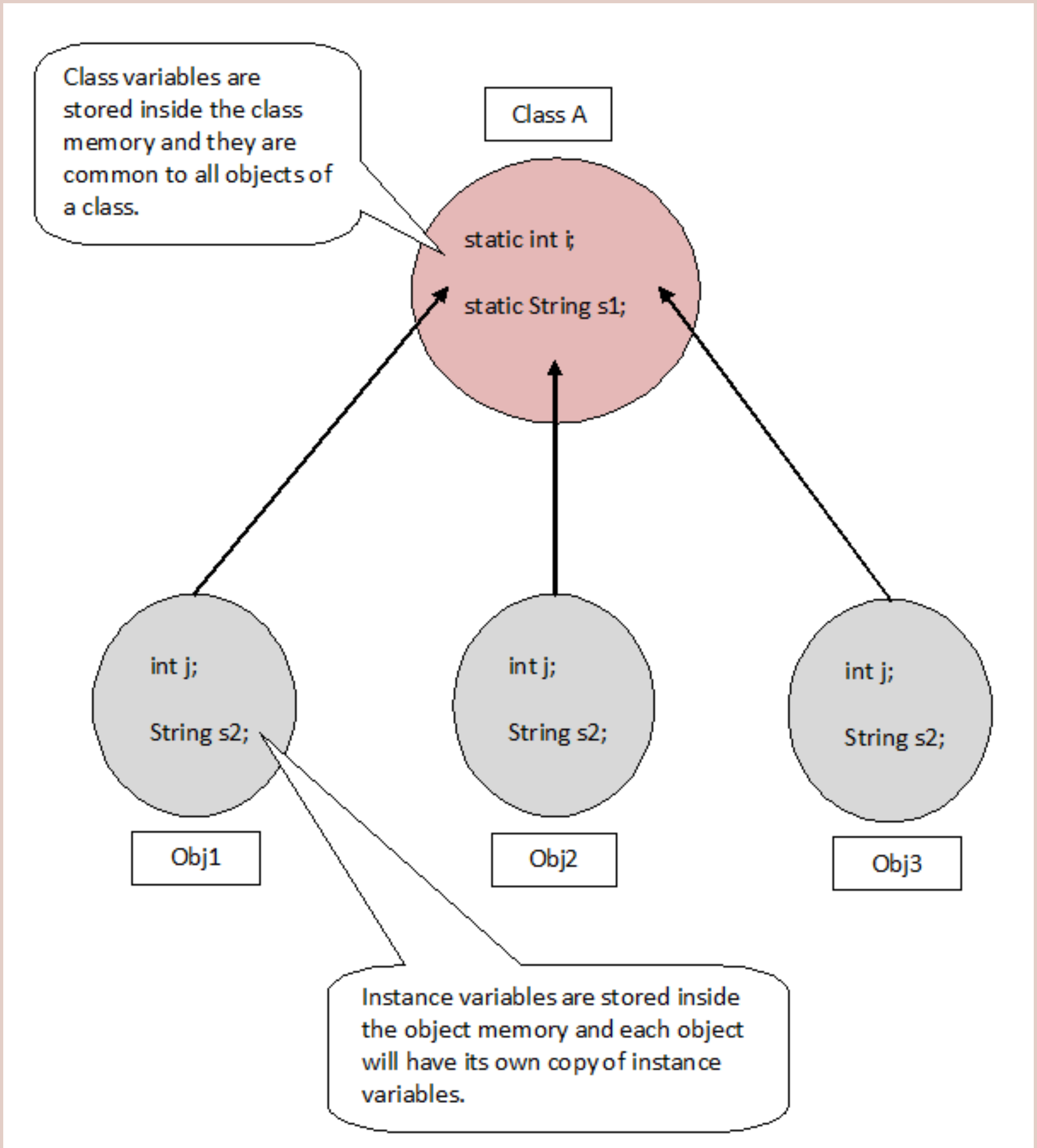
  Should be shared by all objects of a class

- A `static` field called a **class variable** is used in such cases

# Class vs Instance Variables

Blackboard: Week6/Class_Instance_Variables

# Class vs Instance Variables (cont.)


Class variables are stored inside the class memory and they are common to all objects of a class.
Class A
static int i;
static String s1;
int j;
String s2;
int j;
String s2;
int j;
String s2;
Obj1
Obj2
Obj3
Instance variables are stored inside the object memory and each object will have its own copy of instance variables.

## Motivating `static`

- Suppose we want to store a record of all employees of a company

  In this case the "employee id" is unique but the "company name" is common for all

- When we create a `static` variable for the company name
  - Only one copy of `static companyName` is created

    Makes the program more efficient i.e., saves memory

## Problem:

Create a class `EmployeeRecord` which stores a record (`employeeId, firstname, lastname, companyName`) of all employees in a company.

## EmployeeRecord:

```
Blackboard: Week6/Class_Instance_Variables
```

## `static` Variable Rules

- A `static` variable can be created

  By creating or referencing any instance of a class

- `Static` class members exist

  Even when no objects of the class exist

Blackboard: Week6/Person

## `static` Variable Rules

- If there are multiple instances of a class

  A static variable of the class will be shared by all instances of that class

- This will result in only *one* copy

## A quick tutorial:

```
Blackboard: Week6/Counter_2
```

# static final Variables

- static final variables are **constants**

```
public class MyClass
{
     public static final int MY_VAR=27;
}
```

- The above code will execute as soon as the class MyClass is loaded

  Before a static method is called, and even before any static variable can be accessed

- MY_VAR is public which means any class can use it

  It is final so the value of this variable can never be changed in the current or in any class

## A quick tutorial:

```
public class MyClass
{
    public static final int MY_VAR;
}
```

What is the Problem?

## `static` Methods in Java

- A static method *belongs to the class*

  Rather than an object of the class

- A `static` method can be invoked without creating an instance/object of a class

  e.g., public static void main (String[] args)

# `static` Methods in Java (cont.)

- A `static` method can access a `static` variable and change the value of it

  <<ClassName>>.<<VariableName>>

- A `static` method can be directly called by using the class name

  <<ClassName>>.<<MethodName>>

## `static` Variable Access

`Blackboard: Week6/StaticVariableAccess`

## `static` Method Restrictions

- They can only call other `static` methods

  They must only access static data

- `super` and `this` keywords cannot be used in a `static` method

## A quick tutorial:

```
Blackboard: Week6/StaticCheck_1
```

# `static` Methods in Java

- You cannot call something that does not exist
- Since you have not yet created an object

  The non-static method does not exist yet

- A `static` method (by definition) always exists

## Solution 1: Create an Object of the Class

```
Blackboard: Week6/StaticCheck_2
```

## Solution 2: Create an Object of the Class

```
Blackboard: Week6/StaticCheck_3
```

## Solution 3: Declare the Method as `static`

`Blackboard: Week6/StaticCheck_4`

## Rule-of-thumb

- Q: “Does it make sense to call this method, even if no object has been constructed yet?”
  - If so, it should be static

- e.g., `Class Car` might have a method `double convertMpgToKpl(double mpg)`

  Which would be static

  Want to know what 35mog converts to, even if nobody has ever built a Car

- `setMileage(double mpg)` which sets the efficiency of one Car

  Cannot be static

  Inconceivable to call the method before any Car has been constructed

## `static` Import

- A `static` import declaration enables you to import the `static` members of a class or an interface

- You can access them via their *unqualified names* in your class

  i.e., the class name and a dot (.) are not required when using an imported static member

## Problem:

Create a class that calculates the `Square Root`, `Ceiling` of a given value and the value of PI to a specified number of decimal places.

Make use of the static java.lang.Math library and call the in-built functions via their unqualified names.

## `static` Import Example

`Blackboard: Week6/StaticImportTest`