# *Advanced Programming*
# CSE 201
## Instructor: Sambuddho

(Semester: Monsoon 2024)
Week 2 – Classes and Objects - Basics

# Classes and Objects.

- Defining your own class:

```
class ClassName
{
   field1
   field2
   . . .
   constructor1
   constructor2
   . . .
   method1
   method2
   . . .
}
```

```
class Employee
{
   // instance fields
   private String name;
   private double salary;
   private LocalDate hireDay;

   // constructor
   public Employee(String n, double s, int year, int month, int day)
   {
      name = n;
      salary = s;
      hireDay = LocalDate.of(year, month, day);
   }

   // a method
   public String getName()
   {
      return name;
   }

   // more methods
   . . .
}
```

# Classes and Objects.

- Defining your own class:
-
- Array of objects:
-

```
Employee[] staff = new Employee[3];

staff[0] = new Employee("Carl Cracker", . . .);
staff[1] = new Employee("Harry Hacker", . . .);
staff[2] = new Employee("Tony Tester", . . .);
```

Class with main() method should *ideally* be different from the class whose objects it uses.

# Classes and Objects.

```
import java.time.*;

/**
 * This program tests the Employee class.
 * @version 1.13 2018-04-10
 * @author Cay Horstmann
 */
public class EmployeeTest
{
   public static void main(String[] args)
   {
      // fill the staff array with three Employee objects
      Employee[] staff = new Employee[3];

      staff[0] = new Employee("Carl Cracker", 75000, 1987, 12, 15);
      staff[1] = new Employee("Harry Hacker", 50000, 1989, 10, 1);
      staff[2] = new Employee("Tony Tester", 40000, 1990, 3, 15);

      // raise everyone's salary by 5%
      for (Employee e : staff)
         e.raiseSalary(5);

      // print out information about all Employee objects
      for (Employee e : staff)
         System.out.println("name=" + e.getName() + ",salary=" + e.getSalary() + ",hireDay="
            + e.getHireDay());
   }
}

class Employee
{
   private String name;
   private double salary;
   private LocalDate hireDay;

   public Employee(String n, double s, int year, int month, int day)
   {
      name = n;
      salary = s;
      hireDay = LocalDate.of(year, month, day);
   }

   public String getName()
   {
      return name;
   }
```

# Private/Public Methods and Variables

- - Unlike C/C++ all methods and variables are public.
- - To make a class visible to the JVM and other files you must declare it as **public**.
- - All public classes need their respective files.
- - Non-public (private) class need not have their own files.

# Private/Public Methods and Variables

- **Final** keyword
- - WORM type field (define and initialize in the beginning and use as many times as you like without reassining or changing it).

- - Can be used for normal variable as well as class object variables.
- **Private** keyword
  - - Private keyword for class members – cannot be accessed through objects. Can only be accessed by class methods.

# Static Methods and Variables

- **Static** keyword

- - Philosophy: Have a single instance of a class object, variable, method *etc.*
- - Class variable with ‘static’ keyword – Can be used without the class object only with the class name only.
- - Only *one* variable shared across all objects of the class.

# Static Methods and Variables

- **Static** constants
- static final

```
public class Math
{
    . . .
    public static final double PI = 3.14159265358979323846;
    . . .
}
```

# Constructors and Constructor Overloading

- - Called when the class object is created.
- - Explicit constructor vs implicit.

- Implicit: Default when there are no explicit constructors.

- Explicit: Function with same name as class and no return values. Can have any number of arguments.

  - - Constructor overloading: Multiple constructors with the same name, differing only wrt the args.

# Intializations

- - Constructor(s).
- - Value initialization at declaration time.
- - Value *initialization block.*

```
class Employee
{
   private static int nextId;

   private int id;
   private String name;
   private double salary;

   // object initialization block
   {
      id = nextId;
      nextId++;
   }

   public Employee(String n, double s)
   {
      name = n;
      salary = s;
   }

   public Employee()
   {
      name = "";
      salary = 0;
   }
   . . .
}
```

- - NO read before
- initilization.

# Packages and Imports

- - Collection of classes – *package*.
- - *Packaged* in a directory.
- - Packages are clustered in a directory structure.
- - Classes with same name can reside inside their respective *packages.* No conflict.
- - Classname can be a fully qualified classname:

    - java.time.LocalDate

    - OR

    - Implicit based on the imported packages:

    - import java.time.*;
    - LocalDate today = LocalDate.now();

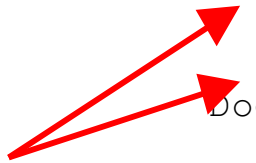
# What is Memory?

- Memory (system memory, not disk or other peripheral devices) is the hardware in which computers store information, both temporary and permanent

- Think of memory as a list of slots; each slot holds information (e.g., a local int variable, or a reference to an instance of a class)

- Here, two references are stored in memory: one to a Dog instance, and one to a DogGroomer instance

```
//Elsewhere in the program
Petshop petSmart = new Petshop();


public class PetShop {


   public PetShop() {
         this.testGroomer();
   }


   public void testGroomer() {
         Dog django = new Dog();
         DogGroomer groomer = new
DogGroomer();
         groomer.groom(django);
   }
```



# Objects as Parameters: Under the Hood (1/6)

```
public class PetShop {

   public PetShop() {
        this.testGroomer();
   }

   public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
   }
```

```
public class DogGroomer {

   public DogGroomer() {
        // this is the constructor!
   }

   public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here
   }
}
```

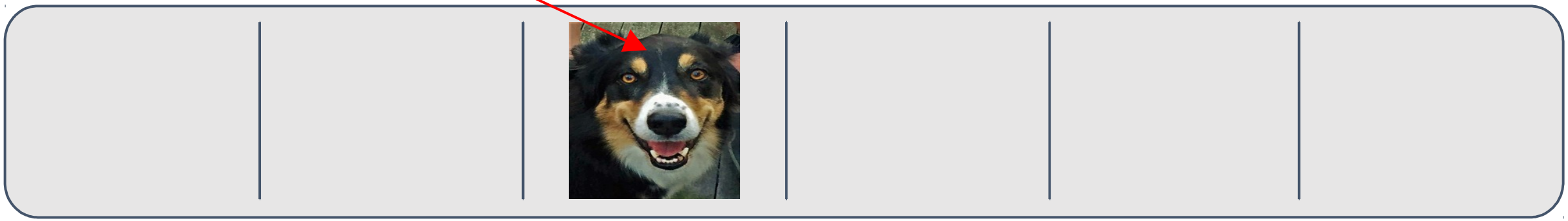
Somewhere in memory...

# Objects as Parameters: Under the Hood (2/6)

```
public class PetShop {

    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
    }
```

```
public class DogGroomer {

    public DogGroomer() {
        // this is the constructor!
    }

    public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here!
    }
}
```

Somewhere in memory...



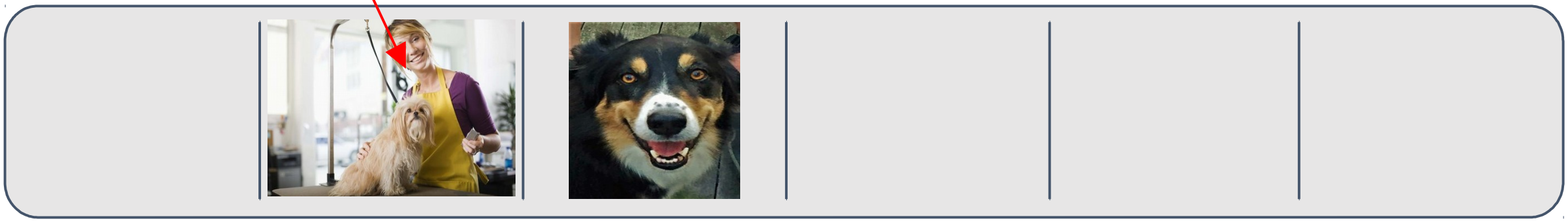
When we instantiate a `Dog`, he's stored somewhere in memory. Our `PetShop` will use the

# Objects as Parameters: Under the Hood (3/6)

```
public class PetShop {

    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
    }
```

```
public class DogGroomer {

    public DogGroomer() {
        // this is the constructor!
    }

    public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here!
    }
}
```

Somewhere in memory...



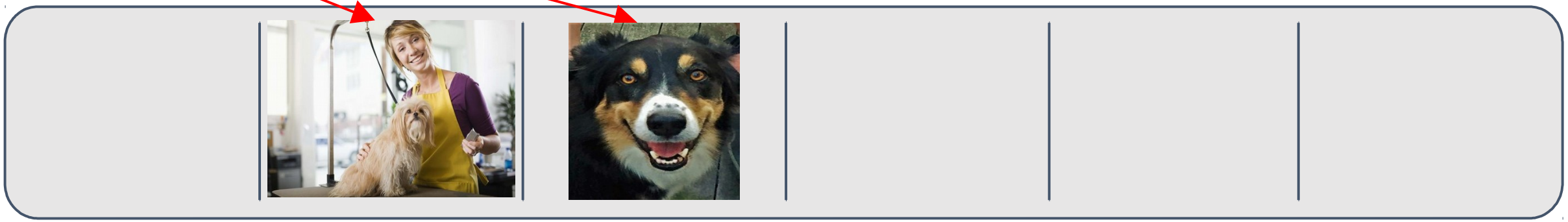
The same goes for the `DogGroomer`—we store a particular `DogGroomer` somewhere in

# Objects as Parameters: Under the Hood (4/6)

```
public class PetShop {

    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
    }
```

```
public class DogGroomer {

    public DogGroomer() {
        // this is the constructor!
    }

    public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here!
    }
}
```

Somewhere in memory...



We call the `groom` method on our `DogGroomer`, `groomer`. We need to tell her which `Dog` to

# Objects as Parameters: Under the Hood (5/6)

```
public class PetShop {

    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
    }
```

```
public class DogGroomer {

    public DogGroomer() {
        // this is the constructor!
    }

    public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here!
    }
}
```

Somewhere in memory...

# Objects as Parameters: Under the Hood (6/6)

```
public class PetShop {

    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
    }
```

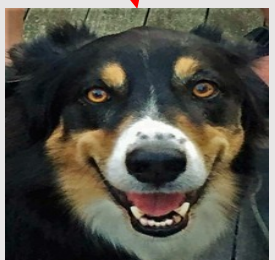
```
public class DogGroomer {

    public DogGroomer() {
        // this is the constructor!
    }

    public void groom(Dog shaggyDog) {
        // code that grooms shaggyDog goes here!
    }
}
```

Somewhere in memory…

The `groom` method doesn't really care which `Dog` it's told to groom—no matter what another

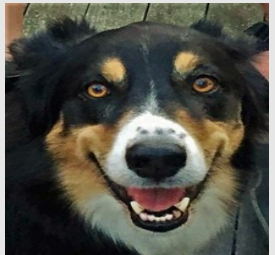
# Variable Reassignment: Under the Hood (1/5)

```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
            this.testGroomer();
    }

    public void testGroomer() {
            Dog django = new Dog();
            DogGroomer groomer = new DogGroomer();
            groomer.groom(django);
            django = new Dog();
            groomer.groom(django);
    }

}
```



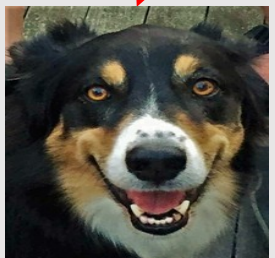
# Variable Reassignment: Under the Hood (2/5)

```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
            this.testGroomer();
    }

    public void testGroomer() {
            Dog django = new Dog();
            DogGroomer groomer = new DogGroomer();
            groomer.groom(django);
            django = new Dog();
            groomer.groom(django);
    }
}
```



# Variable Reassignment: Under the Hood (3/5)

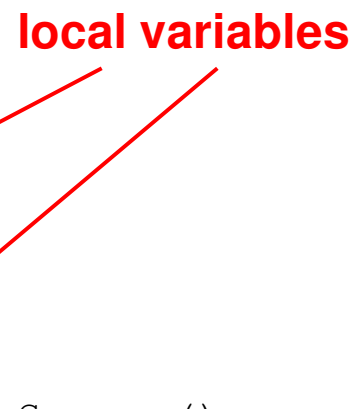
```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
            this.testGroomer();
    }

    public void testGroomer() {
            Dog django = new Dog();
            DogGroomer groomer = new DogGroomer();
            groomer.groom(django);
            django = new Dog();
            groomer.groom(django);
    }
```



# Local Variables (1/2)

- All variables we've seen so far have been **local variables**: variables declared *within a method*
- Problem: the ***scope*** of a local variable (where it is known and can be accessed) is limited to its own method—it cannot be accessed from anywhere else
    - the same is true of method parameters

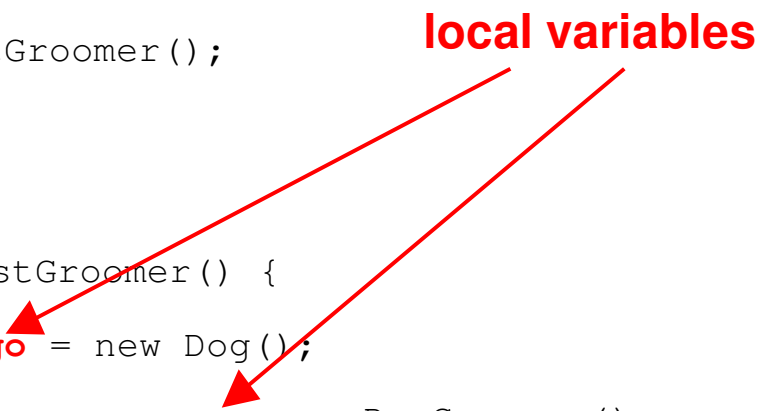
```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
        django = new Dog();
        groomer.groom(django);
    }

}
```



# Local Variables (2/2)

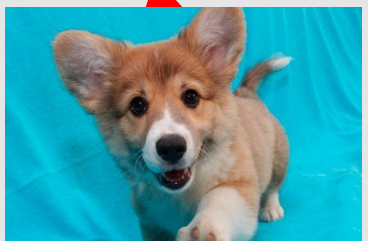
- We created `groomer` and `django` in our `PetShop`'s helper method, but as far as the rest of the class is concerned, they don't exist
- What happens to django after the method is executed?
  - "Garbage Collection"

```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
        django = new Dog();
        groomer.groom(django);
    }

}
```



# Variable Reassignment: Under the Hood (4/5)

```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
            this.testGroomer();
    }

    public void testGroomer() {
            Dog django = new Dog();
            DogGroomer groomer = new DogGroomer();
            groomer.groom(django);
            django = new Dog();//old ref garbage collected
            groomer.groom(django);
    }
```



# Variable Reassignment: Under the Hood (5/5)

```
public class PetShop {

    /* This is the constructor! */
    public PetShop() {
        this.testGroomer();
    }

    public void testGroomer() {
        Dog django = new Dog();
        DogGroomer groomer = new DogGroomer();
        groomer.groom(django);
        django = new Dog();          //old ref garbage collected
        groomer.groom(django);
    }
```