Inheritance

Lecture 5

Dr. Tamer ABUHMED
Java Programming Course (SWE2023)
College of Computing

Outline

- What is Inheritance?
- Subclass and Superclass
- “is-a” vs. “has-a” relationships
- Simple Inheritance Example
- Strategy for Coding with Inheritance
- `protected` Members
- Constructors in Subclasses
- Full Example with Inheritance

What is Inheritance?

- A form of software reuse
- Object-oriented programming (OOP) technique
- Create a new class from an existing class
  - Absorb existing class data (fields) and methods
  - Enhance with new or modified capabilities
- Why do we use Inheritance?
  - Used to eliminate redundant code

How to do inheritance?

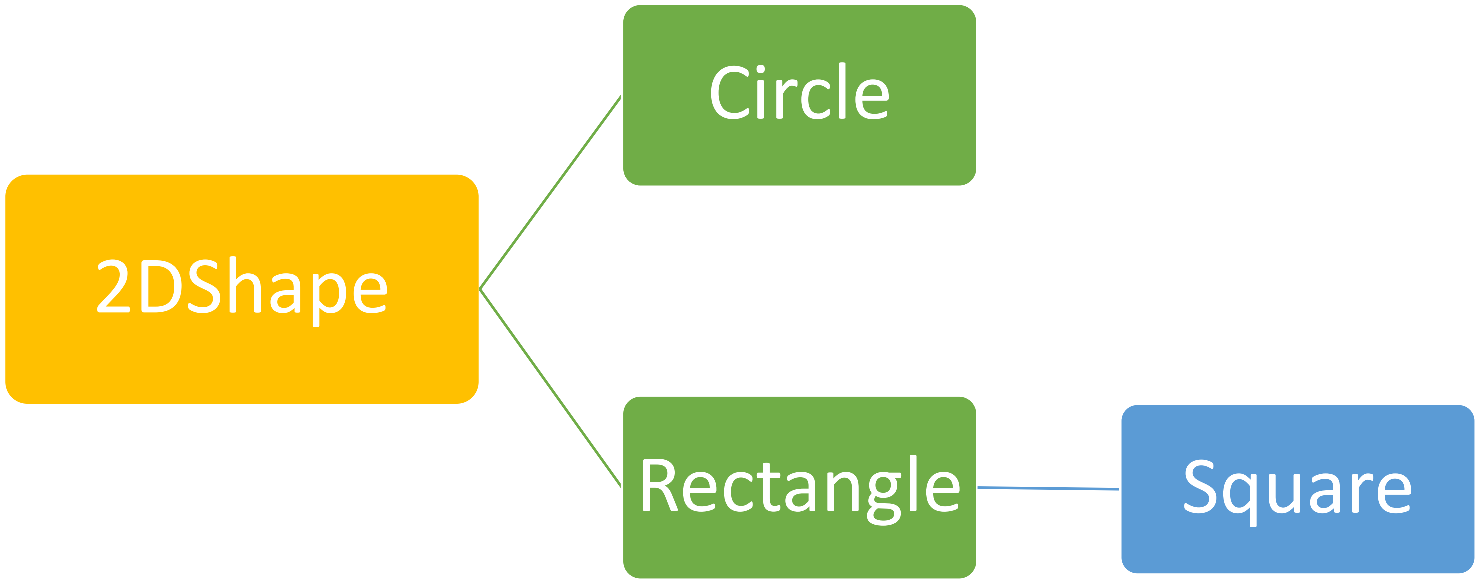
- *With inheritance*, a very general form of a class is first defined, and then more specialized versions of the class are defined
  - The specialized classes are said to *inherit* the methods and instance variables of the general class
- Example
  - Dog class inherits from Animal class
  - Dog *extends* Animal

Subclass and Superclass

- Subclass extends superclass
  - Subclass
    - Also called *child class* or *derived class*
    - More specialized group of objects
    - Inherits data and methods from superclass
    - Can add or modify methods
      - Modifying methods is called *overriding*
  - Superclass
    - Also called *parent class* or *base class*
    - Typically represents larger group of objects
    - Supplies data and behaviors to subclass
    - May be direct or indirect
- Java does not support multiple inheritance

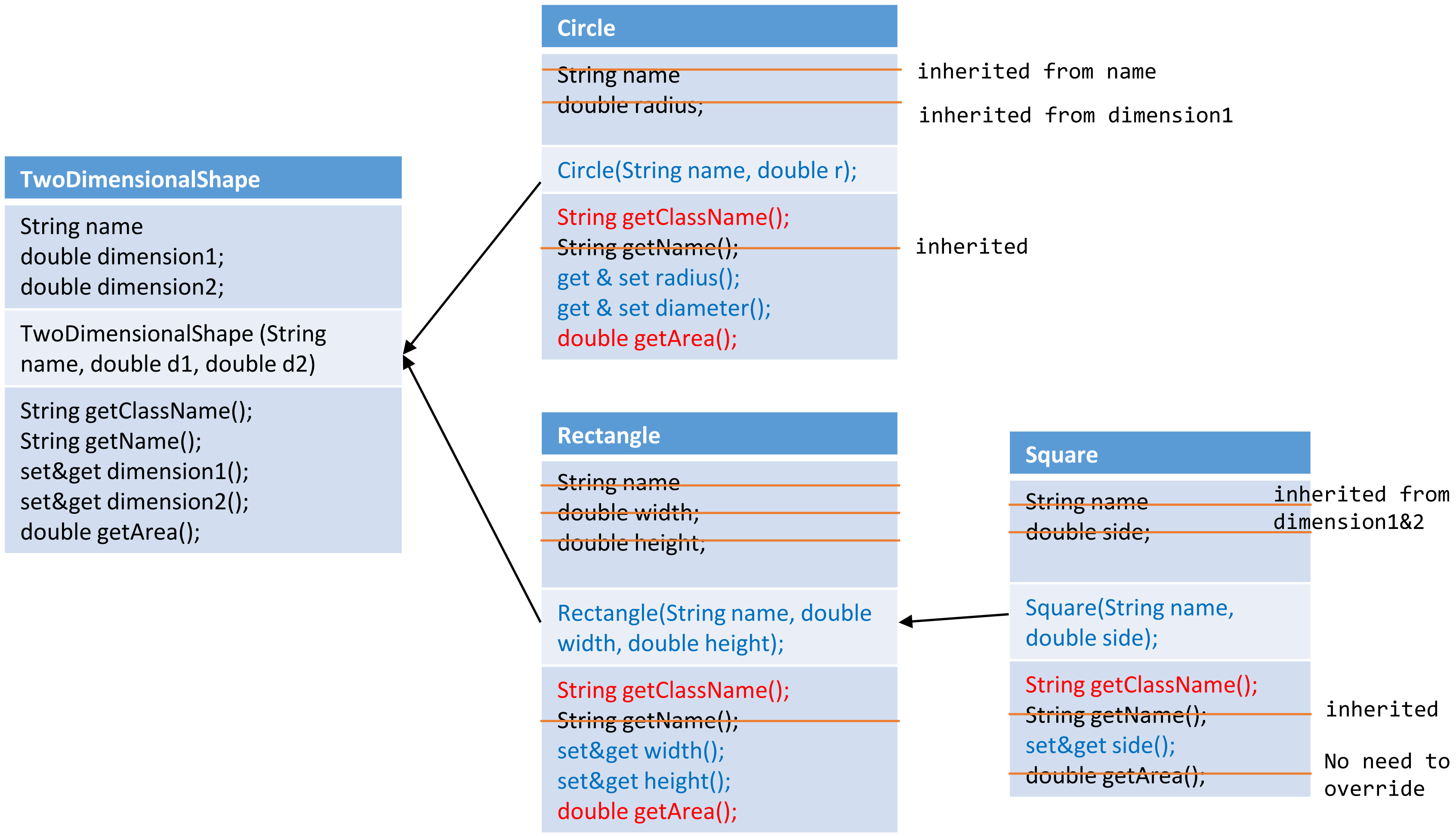
Class hierarchy

- Design superclasses to store common characteristics
- Design the subclasses to store specialized characteristics



Class hierarchy

- Design superclasses to store common characteristics
- Design the subclasses to store specialized characteristics



TwoDimensionalShape

String name
double dimension1;
double dimension2;

TwoDimensionalShape (String name, double d1, double d2)

String getClassName();
String getName();
set&get dimension1();
set&get dimension2();
double getArea();

Circle

~~String name~~
~~double radius;~~

Circle(String name, double r);

String getClassName();
~~String getName();~~
get & set radius();
get & set diameter();
double getArea();

Rectangle

~~String name~~
~~double width;~~
~~double height;~~

Rectangle(String name, double width, double height);

String getClassName();
~~String getName();~~
set&get width();
set&get height();
double getArea();

Square

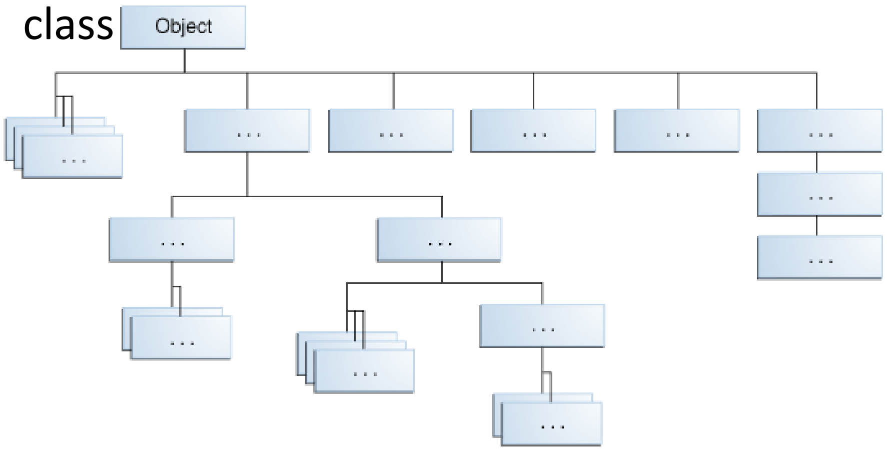
~~String name~~
~~double side;~~

Square(String name, double side);

String getClassName();
~~String getName();~~
set&get side();
~~double getArea();~~

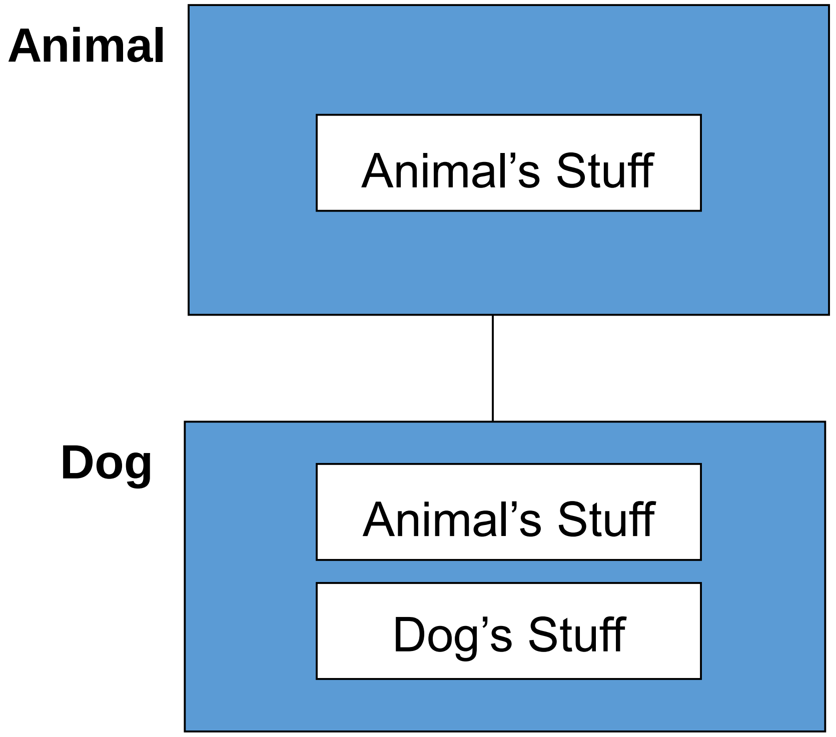
The `Object` class

- Top of the Java class hierarchy
- Located in package `java.lang`
- Class from which every other Java class inherits
- A class implicitly extends Object if no other class is specified
- `.toString(), .clone(), .equals()`



Inheritance in Java

- Dog extends ("is-a") Animal



```
public class ClassName extends SuperClass {
    ...
}
```

```
public class  Dog      extends  Animal   {
    ...
}
```

Simple Inheritance Example (1/2)

```
public class Animal {
        private String name;

        public String getName() {
                return name;
        }

        public void setName(String name)
    {
                this.name = name;
        }

        public String voice() {
                return "?";
        }
}
```

```
class Dog extends Animal {
   public String voice() {
          return "WOOF!";
   }

   public void fetch(String toy) {
          System.out.println("Fetching a " + toy);
          }
}
```

This is an overridden method, Inherited from animal.

This is a new method.

Simple Inheritance Example (2/2)

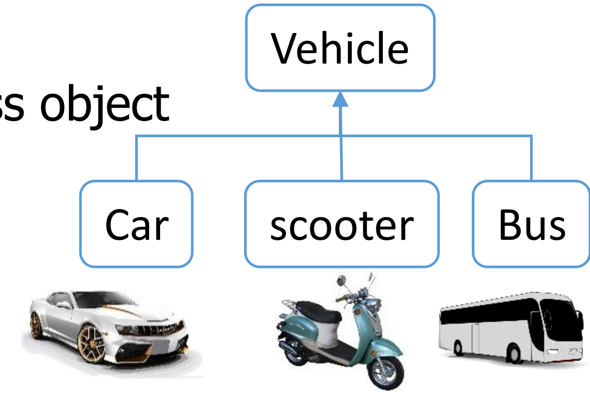
```
public class PetStore
{
  public static void main (String[] args)
        {
        Dog d = new Dog();
              d.setName(“Henry”);
        System.out.println (d.getName() + " says "
        + d.voice());
        }
}
```

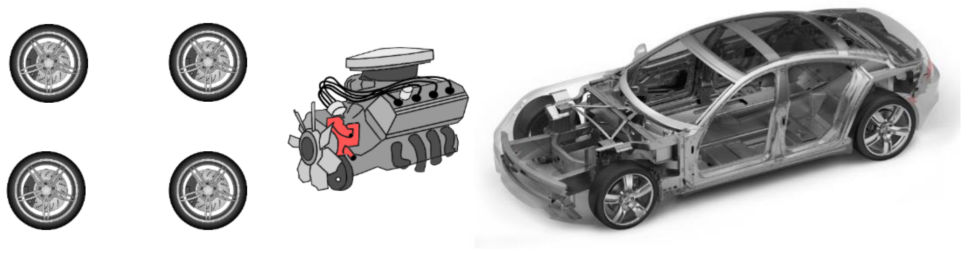
Output

```
Henry says WOOF!
```

“is-a” vs. “has-a” relationships

- “is-a”
  - Represents ***inheritance***
  - subclass object is an example of the superclass object
  - Example: a Car *is a* Vehicle
  - Car is subclass; Vehicle is superclass
  - Keywords: extends, implements
- “has-a”
  - Represents ***composition***
  - Object contains one or more objects of other classes as members
  - Example: Car *has a* Steering Wheel





Strategy for Coding with Inheritance

- Design classes for objects
- Identify characteristics classes have in common
  - Abstraction: focus on commonalities among objects in a system
- Design *superclasses* to store common characteristics
- Design the *subclasses* to store specialized characteristics

Inherited Members

What members of the superclass are going to be inherited by the subclass?

Inherited	*Not* Inherited
Public members	Constructors
Protected members	Private methods, and Variables
Protected Variables	

Static methods in **Java** are **inherited**, but can not be overridden

`protected` Members

- Intermediate level of protection between `public` and `private`
- Accessible to
  - superclasses
  - subclasses
  - classes in the same package
- Use super. to access a superclass method that has been overridden by a subclass method
- Recommendation: Don’t use protected instance variables!
  - “Fragile” software can “break” if superclass changes

Access Modifiers in Java

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

When no access modifier is specified for a class , method or data member, then it is **default** access

private: methods or data members declared as private are accessible only **within the class**

protected: methods or data members declared as protected are **accessible within same package or sub classes in different package.**

public : Classes, methods or data members which are declared as public are **accessible from every where**

Constructors in Subclasses

- Constructors are *not* inherited!
- Chain of constructor calls
  - subclass constructor invokes superclass constructor
    - Implicitly or explicitly
    - To call explicitly, use super()
    - Superclass constructor call must be first statement in subclass constructor
  - `Object` constructor is always fired last
- All instance variables are inherited
  - Private variables not directly accessible

Full Example V1.0 (1/2)

```java
public class CommissionEmployee extends Object
{
   private String firstName;
   private String lastName;
   private String socialSecurityNumber;
   private double grossSales; // gross weekly sales
   private double commissionRate; // commission percentage

   // five-argument constructor
   public CommissionEmployee( String first, String last, String ssn,
      double sales, double rate )
   {
      // implicit call to Object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      setGrossSales( sales ); // validate and store gross sales
      setCommissionRate( rate ); // validate and store commission rat
   } // end five-argument CommissionEmployee constructor

   // set first name
   public void setFirstName( String first )
   {
      firstName = first; // should validate
   } // end method setFirstName

   // return first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

   // set last name
   public void setLastName( String last )
   {
      lastName = last; // should validate
   } // end method setLastName

   // return last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

   // set social security number
   public void setSocialSecurityNumber( String ssn )
   {
      socialSecurityNumber = ssn; // should validate
   }
```

```java
   public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

   // set gross sales amount
   public void setGrossSales( double sales )
   {
      if ( sales >= 0.0 )
         grossSales = sales;
      else
         throw new IllegalArgumentException(
            "Gross sales must be >= 0.0" );
   } // end method setGrossSales

   // return gross sales amount
   public double getGrossSales()
   {
      return grossSales;
   } // end method getGrossSales

   // set commission rate
   public void setCommissionRate( double rate )
   {
      if ( rate > 0.0 && rate < 1.0 )
         commissionRate = rate;
      else
         throw new IllegalArgumentException(
            "Commission rate must be > 0.0 and < 1.0" );
   } // end method setCommissionRate

   // return commission rate
   public double getCommissionRate()
   {
      return commissionRate;
   } // end method getCommissionRate

   // calculate earnings
   public double earnings()
   {
      return commissionRate * grossSales;
   } // end method earnings

   // return String representation of CommissionEmployee object
   @Override
// indicates that this method overrides a superclass method
   public String toString()
   {
      return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s:
         %.2f", "commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales,
         "commission rate", commissionRate );
   } // end method toString } // end class CommissionEmployee
```

Full Example V1.0 (2/2)

```java
public class BasePlusCommissionEmployee extends CommissionEmployee
{
   private double baseSalary; // base salary per week

   // six-argument constructor
   public BasePlusCommissionEmployee( String first, String last,
      String ssn, double sales, double rate, double salary )
   {
      // explicit call to superclass CommissionEmployee constructor
      super( first, last, ssn, sales, rate );

      setBaseSalary( salary ); // validate and store base salary
   } // end six-argument BasePlusCommissionEmployee constructor

   // set base salary
   public void setBaseSalary( double salary )
   {
      if ( salary >= 0.0 )
         baseSalary = salary;
      else
         throw new IllegalArgumentException(
            "Base salary must be >= 0.0" );
   } // end method setBaseSalary

   // return base salary
   public double getBaseSalary()
   {
      return baseSalary;
   } // end method getBaseSalary

   // calculate earnings
   @Override // indicates that this method overrides a superclass method
   public double earnings()
   {
      // not allowed: commissionRate and grossSales private in superclass
      return baseSalary + ( commissionRate * grossSales );
   } // end method earnings
```

```java
   // return String representation of
BasePlusCommissionEmployee
   @Override // indicates that this method overrides a
superclass method
   public String toString()
   {
      // not allowed: attempts to access private superclass
members
      return String.format(
         "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
         "base-salaried commission employee", firstName,
lastName, "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate",
commissionRate, "base salary", baseSalary );
   } // end method toString
} // end class BasePlusCommissionEmployee
```

```java
public class BasePlusCommissionEmployeeTest
{
   public static void main( String[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee(
         "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

      // get base-salaried commission employee data
      System.out.println(
         "Employee information obtained by get methods: \n"
);
  System.out.printf( "%s %s\n", "First name is",
         employee.getFirstName() );
  System.out.printf( "%s %s\n", "Last name is",
         employee.getLastName() );
  System.out.printf( "%s %s\n", "Social security number is",
employee.getSocialSecurityNumber() );
      System.out.printf( "%s %.2f\n", "Gross sales is",
         employee.getGrossSales() );
  System.out.printf( "%s %.2f\n", "Commission rate is",
         employee.getCommissionRate() );
  System.out.printf( "%s %.2f\n", "Base salary is",
         employee.getBaseSalary() );

      employee.setBaseSalary( 1000 ); // set base salary

      System.out.printf( "\n%s:\n\n%s\n",
      "Updated employee information obtained by toString",
         employee.toString() );
   } // end main
} // end class BasePlusCommissionEmployeeTest
```

Compilation Errors

Output

```
Employee information obtained by get methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field CommissionEmployee.firstName is not visible
        The field CommissionEmployee.lastName is not visible
        The field CommissionEmployee.socialSecurityNumber is not visible
        The field CommissionEmployee.grossSales is not visible
        The field CommissionEmployee.commissionRate is not visible

at BasePlusCommissionEmployee.toString(BasePlusCommissionEmployee.java:49)
at BasePlusCommissionEmployeeTest.main(BasePlusCommissionEmployeeTest.java:33)
```

Compilation Errors

```
BasePlusCommissionEmployee.java:39: commissionRate has private access in
CommissionEmployee
      return baseSalary + ( commissionRate * grossSales );
                            ^
BasePlusCommissionEmployee.java:39: grossSales has private access in
CommissionEmployee
      return baseSalary + ( commissionRate * grossSales );
                                             ^
BasePlusCommissionEmployee.java:49: firstName has private access in
CommissionEmployee
         "base-salaried commission employee", firstName, lastName,
                                              ^
BasePlusCommissionEmployee.java:49: lastName has private access in
CommissionEmployee
         "base-salaried commission employee", firstName, lastName,
                                                         ^
BasePlusCommissionEmployee.java:50: socialSecurityNumber has private access
in CommissionEmployee
         "social security number", socialSecurityNumber,
                                   ^
BasePlusCommissionEmployee.java:51: grossSales has private access in
CommissionEmployee
         "gross sales", grossSales, "commission rate", commissionRate,
                        ^
```

```
BasePlusCommissionEmployee.java:51: commissionRate has private access in
CommissionEmployee
         "gross sales", grossSales, "commission rate", commissionRate,
                                                       ^
7 errors
```

Full Example V2.0 (1/2)

```java
public class CommissionEmployee extends Object
{
   protected String firstName;
   protected String lastName;
   protected String socialSecurityNumber;
   protected double grossSales; // gross weekly sales
   protected double commissionRate; // commission percentage

   // five-argument constructor
   public CommissionEmployee( String first, String last, String ssn,
      double sales, double rate )
   {
      // implicit call to Object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      setGrossSales( sales ); // validate and store gross sales
      setCommissionRate( rate ); // validate and store commission rate
   } // end five-argument CommissionEmployee constructor

   // set first name
   public void setFirstName( String first )
   {
      firstName = first; // should validate
   } // end method setFirstName

   // return first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

   // set last name
   public void setLastName( String last )
   {
      lastName = last; // should validate
   } // end method setLastName

   // return last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

   // set social security number
   public void setSocialSecurityNumber( String ssn )
   {
      socialSecurityNumber = ssn; // should validate
   }
```

```java
 public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

   // set gross sales amount
   public void setGrossSales( double sales )
   {
      if ( sales >= 0.0 )
         grossSales = sales;
      else
         throw new IllegalArgumentException(
            "Gross sales must be >= 0.0" );
   } // end method setGrossSales

   // return gross sales amount
   public double getGrossSales()
   {
      return grossSales;
   } // end method getGrossSales

   // set commission rate
   public void setCommissionRate( double rate )
   {
      if ( rate > 0.0 && rate < 1.0 )
         commissionRate = rate;
      else
         throw new IllegalArgumentException(
            "Commission rate must be > 0.0 and < 1.0" );
   } // end method setCommissionRate

   // return commission rate
   public double getCommissionRate()
   {
      return commissionRate;
   } // end method getCommissionRate

   // calculate earnings
   public double earnings()
   {
      return commissionRate * grossSales;
   } // end method earnings

   // return String representation of CommissionEmployee object
   @Override
// indicates that this method overrides a superclass method
   public String toString()
   {
      return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
         "commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales,
         "commission rate", commissionRate );
   } // end method toString
} // end class CommissionEmployee
```

Full Example V2.0 (2/2)

```
public class BasePlusCommissionEmployee extends CommissionEmployee
{
   private double baseSalary; // base salary per week

   // six-argument constructor
   public BasePlusCommissionEmployee( String first, String last,
      String ssn, double sales, double rate, double salary )
   {
      // explicit call to superclass CommissionEmployee constructor
      super( first, last, ssn, sales, rate );

      setBaseSalary( salary ); // validate and store base salary
   } // end six-argument BasePlusCommissionEmployee constructor

   // set base salary
   public void setBaseSalary( double salary )
   {
      if ( salary >= 0.0 )
         baseSalary = salary;
      else
         throw new IllegalArgumentException(
            "Base salary must be >= 0.0" );
   } // end method setBaseSalary

   // return base salary
   public double getBaseSalary()
   {
      return baseSalary;
   } // end method getBaseSalary

   // calculate earnings
   @Override // indicates that this method overrides a superclass method
   public double earnings()
   {
      // not allowed: commissionRate and grossSales private in superclass
      return baseSalary + ( commissionRate * grossSales );
   } // end method earnings

   // return String representation of BasePlusCommissionEmployee
   @Override // indicates that this method overrides a superclass method
   public String toString()
   {
      // not allowed: attempts to access private superclass members
      return String.format(
         "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method toString
} // end class BasePlusCommissionEmployee
```

```
public class BasePlusCommissionEmployeeTest
{
   public static void main( String[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee(
         "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

      // get base-salaried commission employee data
      System.out.println(
         "Employee information obtained by get methods: \n" );
      System.out.printf( "%s %s\n", "First name is",
         employee.getFirstName() );
      System.out.printf( "%s %s\n", "Last name is",
         employee.getLastName() );
      System.out.printf( "%s %s\n", "Social security number is",
         employee.getSocialSecurityNumber() );
      System.out.printf( "%s %.2f\n", "Gross sales is",
         employee.getGrossSales() );
      System.out.printf( "%s %.2f\n", "Commission rate is",
         employee.getCommissionRate() );
      System.out.printf( "%s %.2f\n", "Base salary is",
         employee.getBaseSalary() );

      employee.setBaseSalary( 1000 ); // set base salary

      System.out.printf( "\n%s:\n\n%s\n",
         "Updated employee information obtained by toString",
         employee.toString() );
   } // end main
} // end class BasePlusCommissionEmployeeTest
```

Example V2.0 Output

Output

```
Employee information obtained by get methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00
```

Full Example V3.0 (1/2)

```java
public class CommissionEmployee
 {
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;
    private double grossSales; // gross weekly sales
    private double commissionRate; // commission percentage

    // five-argument constructor
public CommissionEmployee( String first, String last, String ssn,
       double sales, double rate )
    {
       // implicit call to Object constructor occurs here
       firstName = first;
       lastName = last;
       socialSecurityNumber = ssn;
       setGrossSales( sales ); // validate and store gross sales
       setCommissionRate( rate ); }
    // set first name
    public void setFirstName( String first )
    {
       firstName = first; // should validate
    } // end method setFirstName

    // return first name
    public String getFirstName()
    {
       return firstName;
    } // end method getFirstName

    // set last name
    public void setLastName( String last )
    {
       lastName = last; // should validate
    } // end method setLastName

    // return last name
    public String getLastName()
    {
       return lastName;
    } // end method getLastName
```

```java
public void setSocialSecurityNumber( String ssn )
   {
      socialSecurityNumber = ssn; // should validate    }
public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

public void setGrossSales( double sales )
   {
      if ( sales >= 0.0 )
         grossSales = sales;
      else
         throw new IllegalArgumentException(
            "Gross sales must be >= 0.0" );      }
public double getGrossSales()
   {
      return grossSales;
   } // end method getGrossSales

public void setCommissionRate( double rate )
   {
      if ( rate > 0.0 && rate < 1.0 )
         commissionRate = rate;
      else
         throw new IllegalArgumentException(
            "Commission rate must be > 0.0 and < 1.0" );
   } // end method setCommissionRate

   public double getCommissionRate()
   {    return commissionRate;    }
public double earnings()
   {
      return getCommissionRate() * getGrossSales();
   } // end method earnings

@Override // indicates that this method overrides a superclass
method
   public String toString()
   {
      return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
         "commission employee", getFirstName(), getLastName(),
         "social security number", getSocialSecurityNumber(),
         "gross sales", getGrossSales(),
         "commission rate", getCommissionRate() );
   } // end method toString
} // end class CommissionEmployee
```

Full Example V3.0 (2/2)

```
public class BasePlusCommissionEmployee extends CommissionEmployee
{
   private double baseSalary; // base salary per week

   // six-argument constructor
   public BasePlusCommissionEmployee( String first, String last,
      String ssn, double sales, double rate, double salary )
   {
      super( first, last, ssn, sales, rate );
      setBaseSalary( salary ); // validate and store base salary
   } // end six-argument BasePlusCommissionEmployee constructor

   // set base salary
   public void setBaseSalary( double salary )
   {
      if ( salary >= 0.0 )
         baseSalary = salary;
      else
         throw new IllegalArgumentException(
            "Base salary must be >= 0.0" );
   } // end method setBaseSalary

   // return base salary
   public double getBaseSalary()
   {
      return baseSalary;
   } // end method getBaseSalary

   // calculate earnings
   @Override // indicates that this method overrides a superclass method
   public double earnings()
   {
      return getBaseSalary() + super.earnings();
   } // end method earnings

   // return String representation of BasePlusCommissionEmployee
   @Override // indicates that this method overrides a superclass method
   public String toString()
   {
      return String.format( "%s %s\n%s: %.2f", "base-salaried",
         super.toString(), "base salary", getBaseSalary() );
   } // end method toString
} // end class BasePlusCommissionEmployee
```

```
public class BasePlusCommissionEmployeeTest
{
   public static void main( String[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee(
         "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

      // get base-salaried commission employee data
      System.out.println(
         "Employee information obtained by get methods: \n" );
      System.out.printf( "%s %s\n", "First name is",
         employee.getFirstName() );
      System.out.printf( "%s %s\n", "Last name is",
         employee.getLastName() );
      System.out.printf( "%s %s\n", "Social security number is",
         employee.getSocialSecurityNumber() );
      System.out.printf( "%s %.2f\n", "Gross sales is",
         employee.getGrossSales() );
      System.out.printf( "%s %.2f\n", "Commission rate is",
         employee.getCommissionRate() );
      System.out.printf( "%s %.2f\n", "Base salary is",
         employee.getBaseSalary() );

      employee.setBaseSalary( 1000 ); // set base salary

      System.out.printf( "\n%s:\n\n%s\n",
         "Updated employee information obtained by toString",
         employee.toString() );
   } // end main
} // end class BasePlusCommissionEmployeeTest
```

Example V3.0 Output

Output

```
Employee information obtained by get methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00
```

Summary

- What is Inheritance?
- Subclass and Superclass
- “is-a” vs. “has-a” relationships
- Simple Inheritance Example
- Strategy for Coding with Inheritance
- `protected` Members
- Constructors in Subclasses
- Full Example with Inheritance

CODING CONVENTION

Code Review Checklist

- ✓ Coding standards
- ☐ Coding Best practices
- ✓ Non Functional Requirements
- ✓ OOAD Principles
- ✓ Static Code Analysis Metrics
- ☐ ....

Coding Convention (What & Why)

- Coding Convention is collection of rules lead to greater consistency within your code and the code of your teammates.
    - makes maintenance of your code a lot easier
    - improve the readability
    - reduce training management and effort
    - avoid junior mistakes.
    - result in a correct entered JavaDoc output
- Different places where the Conventions can be applied
    - Naming Conventions
    - Comments Conventions

Any code is 20% of its time is written and 80% time is read, so write it well

Coding Convention

- Rules that pertain to how code is to be written, including:
  - **File organization**: how code is distributed between files, and organized within each file.
  - **Indentation**: how particular syntactical elements are to be indented in order to maximize readability.
  - **Comments**: how to consistently and efficiently use comments to help program understandability.
  - **Declarations**: what particular syntax to use to declare variables, data structures, classes, etc. in order to maximize code readability.
  - **Naming**: how to give names to various named entities in a program as to convey meaning embedded into the names.

Naming Conventions

- WRONG
  - `public class _HelloWorld{ }`
  - `void PRINT(){`
- RIGHT
  - `public class HelloWorld { }`
  - `void printName(){`

- **Class names**
  - should be nouns,
  - in mixed case with the first letter of each internal word capitalized. Also known as the CamelNotation.
- **Method name**
  - should be verb
  - in mixed case with the first letter lowercase, with the first letter of each internal word capitalized

Naming Conventions (2)

- WRONG
  - `int AMOUNT = 100;`
  - `public static final int heightX = 100;`
  - `package learning.com.java.algorithms._functions;`
- RIGHT
  - `int amount = 100;`
  - `public static final int HEIGHT_X = 100;`
  - `package learning.com.programs.algorithms.functions;`

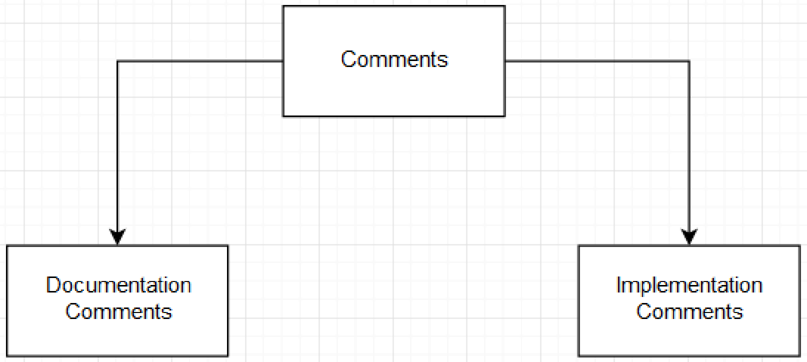
- **Variables**
  - should be short yet meaningful.
  - Non final-name start with a lower-case letter and internal words start with capital letters.
- **Constant**
  - Constant of should contain only upper-case letters and underscores.

Assignment Conventions (3)

- WRONG
  - `fooBar.fChar = barFoo.lchar = 'c’;`
  - `d = (a = b + c) + r;`
- RIGHT
  - `fooBar.fChar = 'c';`
  - `barFoo.lchar = 'c’;`
  - `a = b + c;`
  - `d = a + r;`

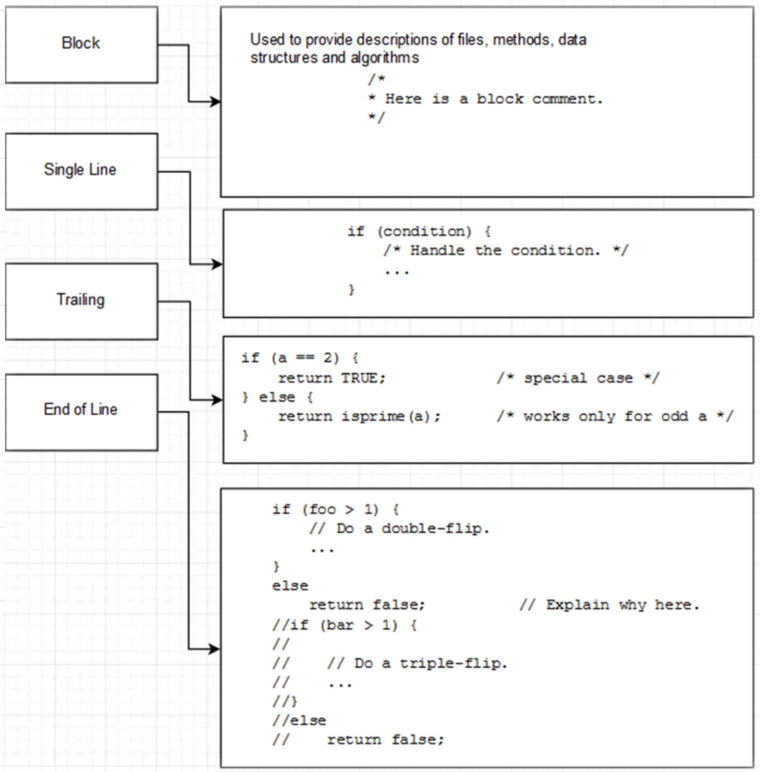
- Avoid assigning several variables to the same value in a single statement. It is hard to read.

Comment Conventions



- Delimited by /**......*/
- Found in both C++ and Java
- Comments about the implementation logic
- Commented codes

- Delimited by /*......*/ and //
- Found only in Java
- Can be extracted to HTML pages using javadoc tools
- Describes the specification of the code
- API documentation
- Will be used by third party developers



Used to provide descriptions of files, methods, data structures and algorithms

```
/*
 * Here is a block comment.
 */
```

```
if (condition) {
    /* Handle the condition. */
    ...
}
```

```
if (a == 2) {
    return TRUE;            /* special case */
} else {
    return isprime(a);      /* works only for odd a */
}
```

```
if (foo > 1) {
    // Do a double-flip.
    ...
}
else
    return false;           // Explain why here.
//if (bar > 1) {
//
//    // Do a triple-flip.
//    ...
//}
//else
//    return false;
```

Comment Conventions

```java
/*
 * Copyright notice
 */

package lab3;

/**
 * class description
 * @version 1.10 04 March 2014
 * @author First name Last name
 */
public class Student {
 /* A class implementation comment can go here. */

 /**
  * class variables - doc comment
  */
 private int stdId;

 /**
  * instance variables - doc comment
  */
 public String stdName;
```

Beginning Comments

Class/interface documentation comment (/**...*/)

Class/interface implementation comment (/*...*/), if necessary

Comment Conventions (2)

```java
     * default constructor
     */
    public Student() {
        stdId = 7;
        stdName = "Ronaldo";

    /**
     * two argument constructor
     * @param colorVariant comment for parameter 1
     * @param colorCode comment for parameter 2
     */
    public Student(int studentId, String studentName) {
        this.stdId = studentId;
        this.stdName = studentName;
    }

    /**
     * @return the student identity
     */
    public int getStudentId() {
        return stdId;
    }

    /**
     * @param studentId student identity
     */
    public void setStudentId(int studentId) {
        stdId = studentId; //inline comment here
}
```

Indentation

Documentation comments

Blank line

Statements Conventions

Simple

Each line should contain at most one stateme
argv++; argc–;   // AVOD

return

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

if-else

```
if (condition){
    statements;
} else if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statement;
}

if(condition)
    statement; //AVOID! THIS OMITS THE BRACES {} !
```

for

```
for (initialization; condition; update) {
    statements;
}
```

while

```
while (condition) {
    statements;
}
```

do-while

```
do {
    statements;
} while (condition)
```

switch

Every time a case falls through (doesn’t include a break statement), add a comment where the break statement would normally be.
Every switch statement should include a default case.

```
switch (condition) {
   case ABC:
      statements;
      /* falls through */
   case DEF:
      statements;
      break;
   case XYZ:
      statements;
      break;
   default:
      statements;
      break;
}
```

try-catch

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

Summary

- Coding Convention is collection of rules lead to greater consistency within your code
  - **File organization**: how code is distributed between files, and organized within each file.
  - **Indentation**: how particular syntactical elements are to be indented in order to maximize readability.
  - **Comments**: how to consistently and efficiently use comments to help program understandability.
  - **Declarations**: what particular syntax to use to declare variables, data structures, classes, etc. in order to maximize code readability.
  - **Naming**: how to give names to various named entities in a program as to convey meaning embedded into the names.