CIS 351-Data Structure-Inheritance, Interface, Polymorphism
Feb 18, 2020

Dr. Farzana Rahman

Syracuse University

Terminology

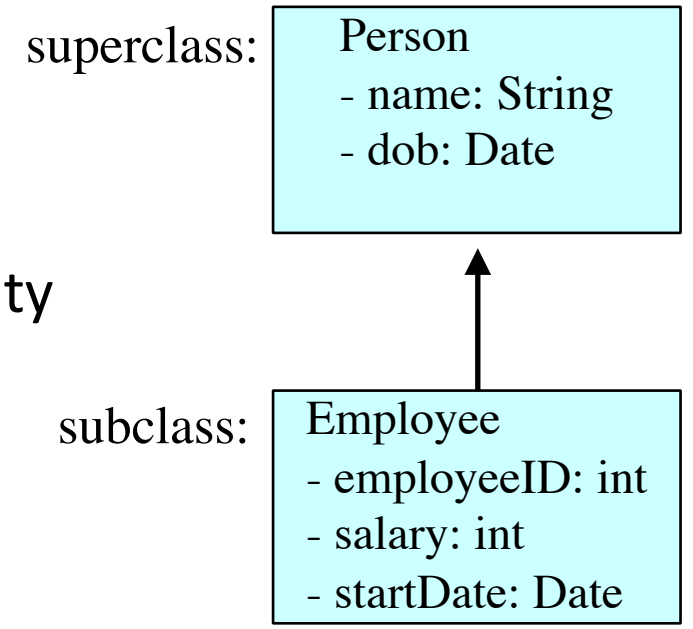
- Inheritance is a fundamental Object Oriented concept

A class can be defined as a "subclass" of another class.

- The subclass inherits all data attributes of its superclass
- The subclass inherits all methods of its superclass
- The subclass inherits all associations of its superclass

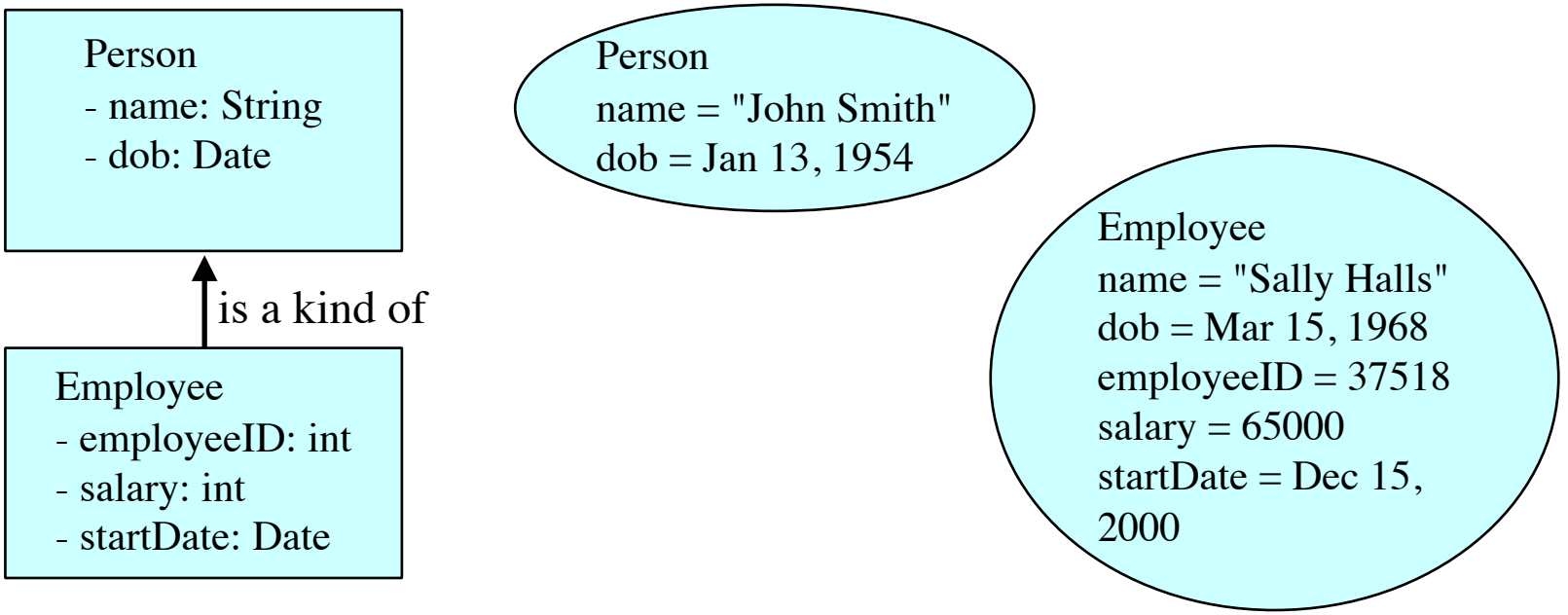
The subclass can:

- Add new functionality
- Use inherited functionality
- Override inherited functionality



What really happens?

- In this example, we can say that an Employee "is a kind of" Person.
  - An Employee object inherits all of the attributes, methods and associations of Person



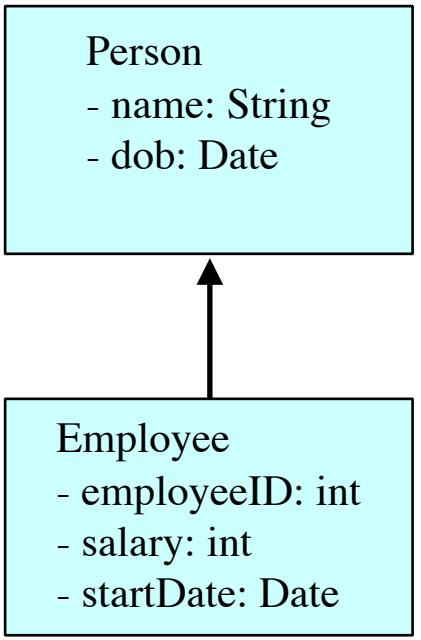
Inheritance in Java

- Inheritance is declared using the "extends" keyword
  - If inheritance is not defined, the class extends a class called Object

```
public class Person
{
  private String name;
  private Date dob;
  [...]
```

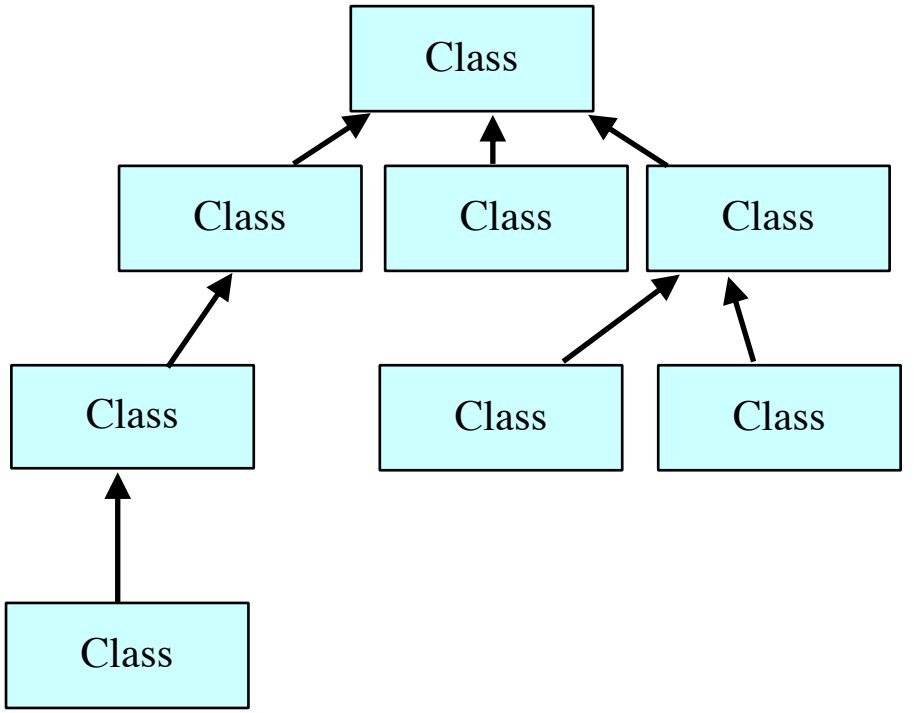
```
public class Employee extends Person
{
  private int employeID;
  private int salary;
  private Date startDate;
  [...]
```

```
Employee anEmployee = new Employee();
```



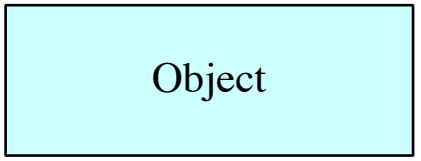
Inheritance Hierarchy

- Each Java class has one (and only one) superclass.
- Inheritance creates a class hierarchy
  - Classes higher in the hierarchy are more general and more abstract
  - Classes lower in the hierarchy are more specific and concrete
- There is no limit to the number of subclasses a class can have
- There is no limit to the depth of the class tree.



The class called Object

- At the very top of the inheritance tree is a class called Object
- All Java classes inherit from Object.
- The Object class is defined in the java.lang package
  - Examine it in the Java API Specification



Constructors and Initialization

- Classes use constructors to initialize instance variables
  - When a subclass object is created, its constructor is called.
  - It is the responsibility of the subclass constructor to invoke the appropriate superclass constructors so that the instance variables defined in the superclass are properly initialized

- Superclass constructors can be called using the "super"
  - It must be the first line of code in the constructor

- If a call to super is not made, the system will automatically attempt to invoke the no-argument constructor of the superclass.

Constructors - Example

```
public class BankAccount
{
  private String ownersName;
  private int accountNumber;
  private float balance;

  public BankAccount(int anAccountNumber, String aName)
  {
      accountNumber = anAccountNumber;
      ownersName = aName;
  }
  [...]
}
public class OverdraftAccount extends BankAccount
{
  private float overdraftLimit;

  public OverdraftAccount(int anAccountNumber, String
  aName, float aLimit)
  {
      super(anAccountNumber, aName);
      overdraftLimit = aLimit;
  }
}
```

Method Overriding

- Subclasses inherit all methods from their superclass
  - Sometimes, the implementation of the method in the superclass does not provide the functionality required by the subclass.
  - In these cases, the method must be overridden.

- To override a method, provide an implementation in the subclass.
  - The method in the subclass MUST have the exact same signature as the method it is overriding.

Method overriding

```
public class BankAccount
{
  private String ownersName;
  private int accountNumber;
  protected float balance;

  public void deposit(float anAmount)
  {
      if (anAmount>0.0)
           balance = balance + anAmount;
  }

  public void withdraw(float anAmount)
  {
      if ((anAmount>0.0) && (balance>anAmount))
           balance = balance - anAmount;
  }

  public float getBalance()
  {
      return balance;
  }
}
```

Method overriding - Example

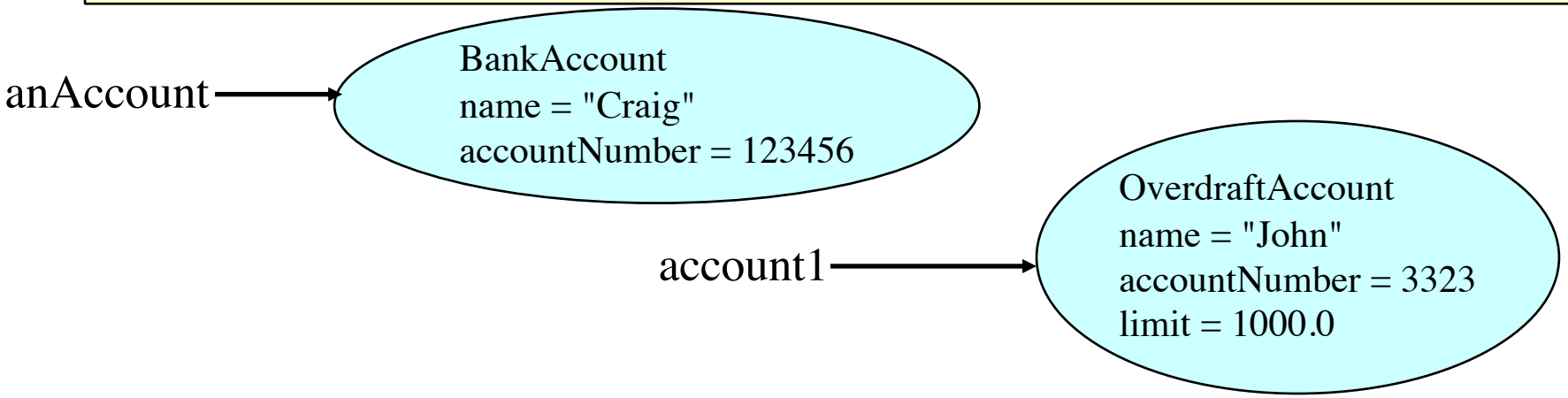
```
public class OverdraftAccount extends BankAccount
{
  private float limit;

  public void withdraw(float anAmount)
  {
      if ((anAmount>0.0) && (getBalance()+limit>anAmount))
           balance = balance - anAmount;
  }

}
```

Object References and Inheritance

- Inheritance defines "a kind of" relationship.
  - In the previous example, OverdraftAccount "is a kind of" BankAccount

- Programmers can "substitute" object references.
  - A superclass reference can refer to an instance of the superclass OR an instance of ANY class which inherits from the superclass.

```
BankAccount anAccount = new BankAccount(123456, "Craig");

BankAccount account1 = new OverdraftAccount(3323, "John", 1000.0);
```



Polymorphism

- In the previous slide, the two variables are defined to have the same type at compile time: BankAccount
  - the types of objects they are referring to at runtime are different

- What happens when the withdraw method is invoked on each object?
  - anAccount refers to an instance of BankAccount. Therefore, the withdraw method defined in BankAccount is invoked.
  - account1 refers to an instance of OverdraftAccount. Therefore, the withdraw method defined in OverdraftAccount is invoked.

- Polymorphism is: The method being invoked on an object is determined AT RUNTIME and is based on the type of the object receiving the message.

Polymorphism

- The term *polymorphism* literally means "having many forms"
- A *polymorphic reference* is a variable that can refer to different types of objects at different points in time
- All object references in Java are potentially polymorphic and can refer to an object of any type compatible with its defined type
- Compatibility of class types can be based on either Inheritance or **Interfaces**
- Polymorphism enables programmers to deal in generalities and
  - let the execution-time environment handle the specifics.

Final Methods and Final Classes

- Methods can be qualified with the final modifier
  - Final methods cannot be overridden.
  - This can be useful for security purposes.

```
public final boolean validatePassword(String username, String
  Password)
{
  [...]
```

- Classes can be qualified with the final modifier
  - The class cannot be extended
  - This can be used to improve performance. Because there can be no subclasses, there will be no polymorphic overhead at runtime.

```
public final class Color
{
  [...]
```

Interface

- Interface is kind of a contract.
- It does not have an actual implementation and hence you can not instantiate an object of Interface itself.
- Method bodies exist only for default methods and static methods.

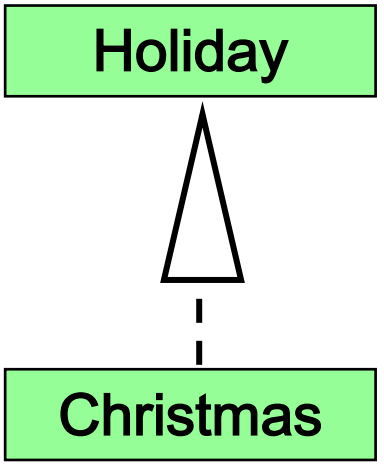
```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

To implement this interface, you'd use the implements keyword in the class declaration:

```
class ACMEBicycle implements Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;

   // The compiler will now require that methods
   // changeCadence, changeGear, speedUp, and applyBrakes
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" + cadence + "
speed:" +
                speed + " gear:" + gear);
    }
}
```

References and Interfaces

- An object reference can refer to an object of its class or to an object of any class related to it by an interface
- For example, if a `Christmas` class implements `Holiday`, then a `Holiday` reference could be used to point to a `Christmas` object




```
Holiday day;
day = new Christmas();
```

Interface Hierarchies

- Inheritance can be applied to interfaces as well as classes
- One interface can be derived from another interface
- The child interface inherits all abstract methods of the parent
- A class implementing the child interface must define all methods from both the ancestor and child interfaces

Visibility Revisited

- All variables and methods of a parent class, even private members, are inherited by its children
- As we've mentioned, private members cannot be referenced by name in the child class
- However, private members inherited by child classes exist and can be referenced indirectly through public methods
- The `super` reference can be used to refer to the parent class, even if no object of the parent class exists
- Allow each class to manage its own data; use the `super` reference to invoke the parent's constructor to set up its data
- Even if there are no current uses for them, override general methods such as `toString` and `equals` with appropriate definitions
- Use visibility modifiers carefully to provide needed access without violating encapsulation

Restricting Inheritance

- The `final` modifier can be used to curtail inheritance
- If the `final` modifier is applied to a method, then that method cannot be overridden in any descendent classes
- If the `final` modifier is applied to an entire class, then that class cannot be used to derive any children at all
- These are key design decisions and establish that a method or class must be used “as is” or not at all