

Inheritance

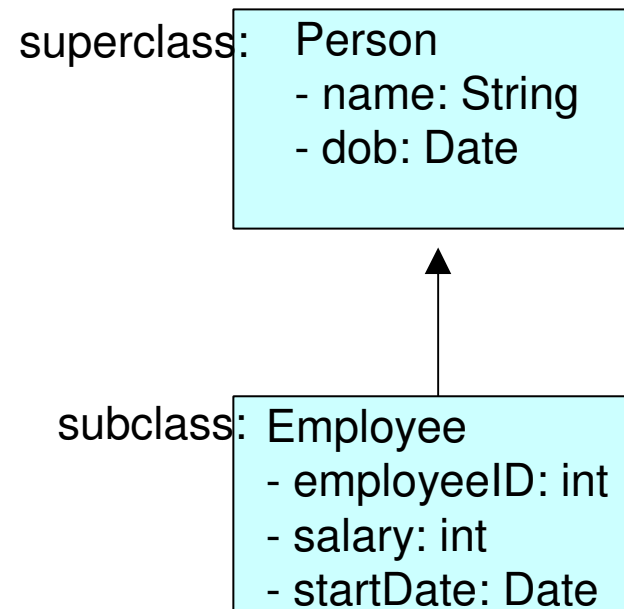


The objectives of this chapter are:

- To explore the concept and implications of inheritance
 - Polymorphism
- To define the syntax of inheritance in Java
- To understand the class hierarchy of Java
- To examine the effect of inheritance on constructors

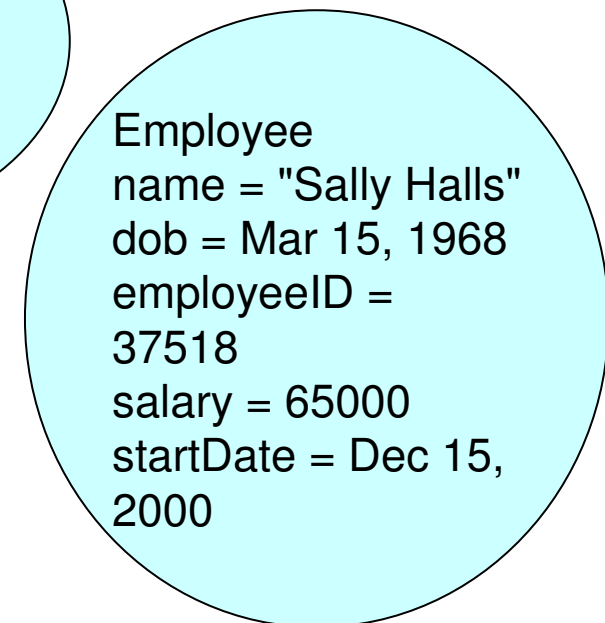
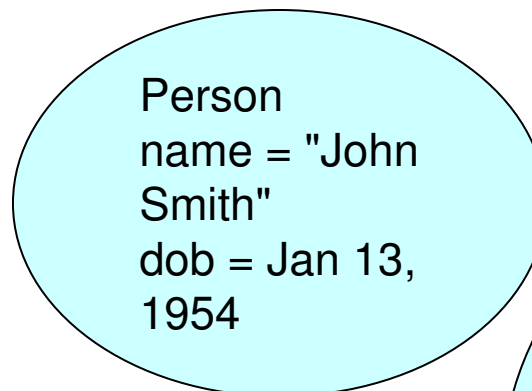
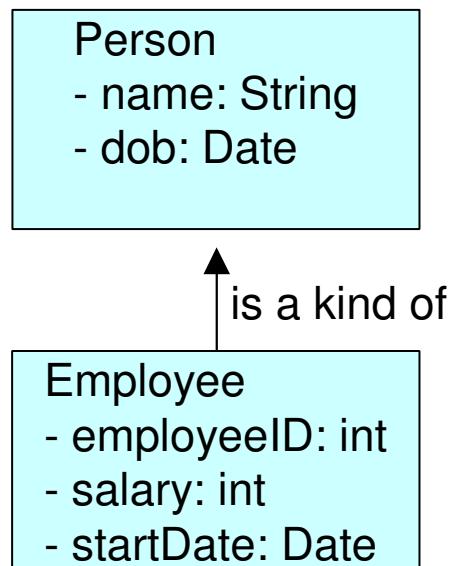
Terminology

- 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?

- When an object is created using new, the system must allocate enough memory to hold all its instance variables.
 - This includes any inherited instance variables
- 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



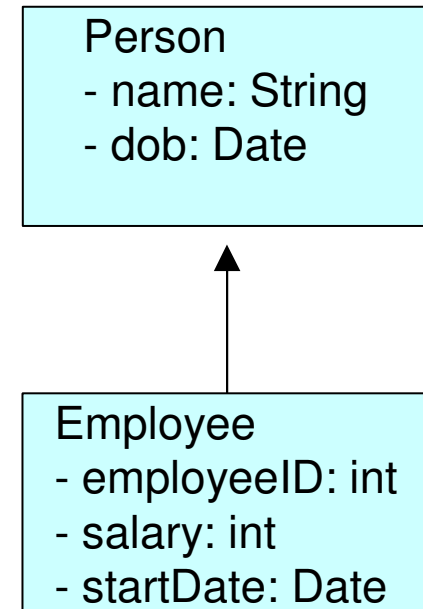
Inheritance in Java

- 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;
    [...]
```

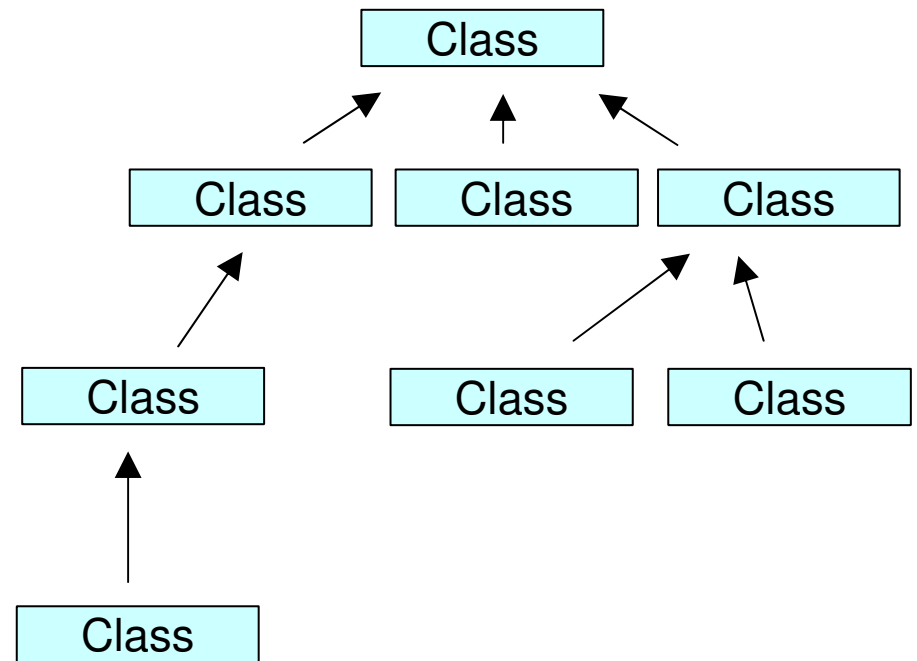
```
public class Employee extends Person
{
    private int employeeID;
    private int salary;
    private Date startDate;
    [...]
```

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



Inheritance Hierarchy

- Each Java class has one (and only one) superclass.
- C++ allows for multiple inheritance
- 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.
 - All objects have a common ancestor
 - This is different from C++
- The Object class is defined in the java.lang package
 - Examine it in the Java API Specification

Object



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" keyword in a manner similar to "this"
 - 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 - Example



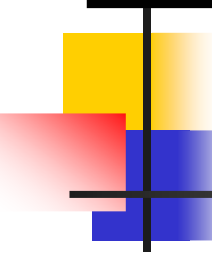
```
public class BankAccount
{
    private String ownersName;
    private int accountNumber;
    protected float balance;

    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;

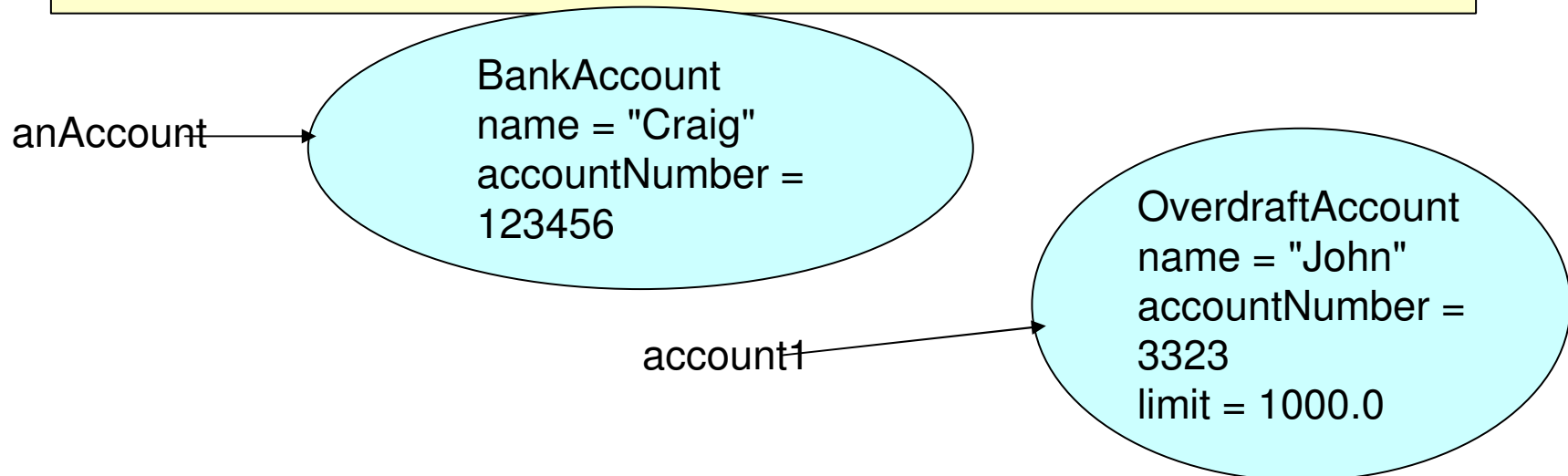
    public void withdraw(float anAmount)
    {
        if ((anAmount>0.0) && (getBalance()+limit>anAmount))
            balance = balance - anAmount;
    }
}
```

Object References and Inheritance

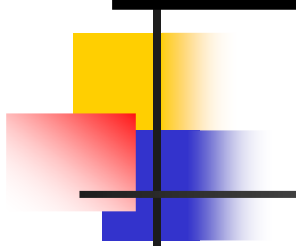
- Inheritance defines "a kind of" relationship.
 - In the previous example, OverdraftAccount "is a kind of" BankAccount
- Because of this relationship, programmers can "substitute" object references.
 - A superclass reference can refer to an instance of the superclass OR an instance of ANY class which 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`
 - However, 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.

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
{
    [...]
}
```

Review

- What is inheritance? What is a superclass? What is a subclass?
- Which class is at the top of the class hierarchy in Java?
- What are the constructor issues surrounding inheritance?
- What is method overriding? What is polymorphism? How are they related?
- What is a final method? What is a final class?