Benefits of Object-Oriented Functionality

- Specify set of **methods** guaranteed to be implemented
- Provide **method implementations and/or data** that get used by multiple related classes
- Define a **group of classes** that can be referred to by the group name
- (all of this within specific limits)

void *

Why did we use them in C?

Why are they dangerous in C?

Tagged unions

Why did we use them in C?

Why are they annoying in C?

Interface

A set of **methods only**

If a class implements an interface, that is a guarantee it has all of that specific set of functions.

Allows a single implementation of a sorted structure (e.g. need to implement a comparison function) to be used for any object that implements **comparable**.

Allows a new kind of type checking

The same class can implement many interfaces

```
public interface Comparable<T>
{
  int compareTo(T o);
}
```

```
public class BinarySearchTree
{
    public void insert( Comparable x );
}
```

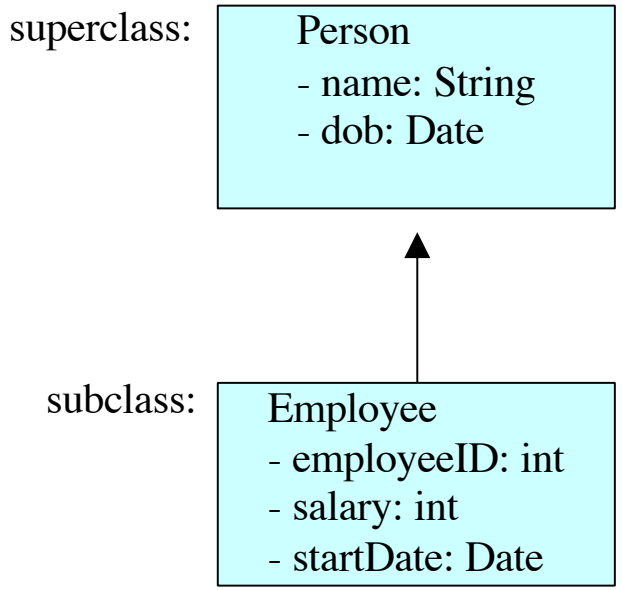
Inheritance

The objectives of this lecture 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

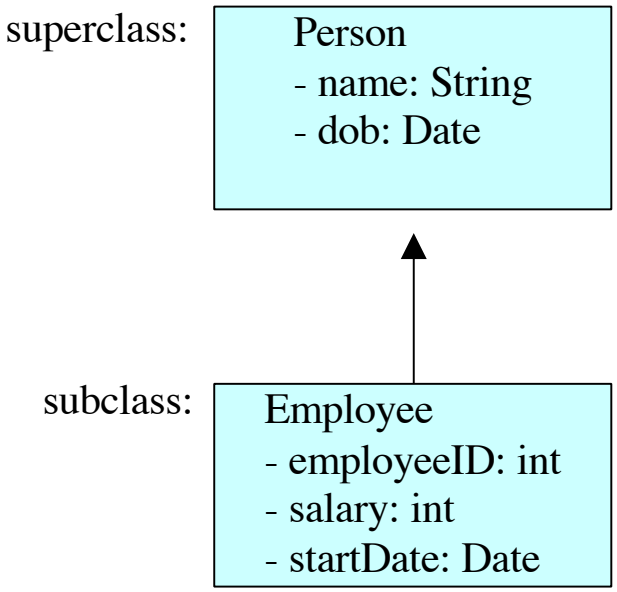
Terminology

- 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



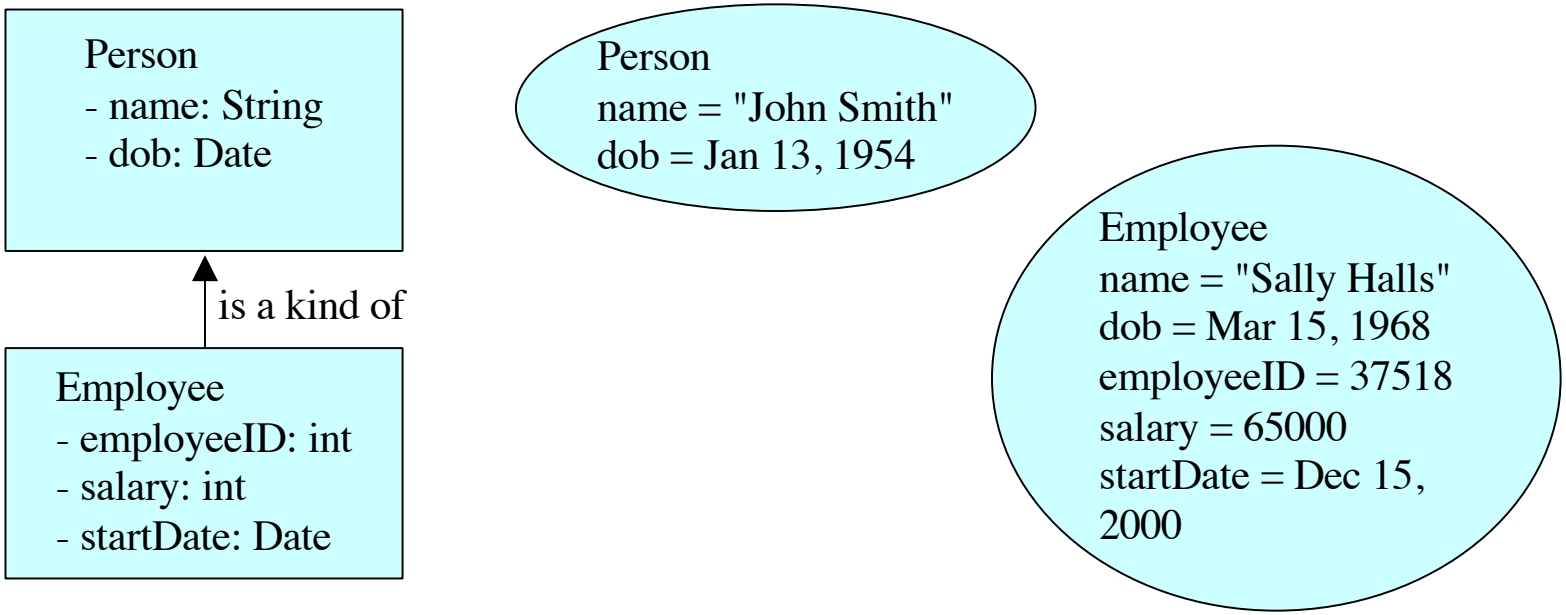
How is this useful?

- Economy of time – When you implement Employee, you already have all functionality of Person. No need to reimplement
- Parameter passing – If you have a function that expects a Person, you can pass an Employee, and it is still fine.
  - Fewer special-purpose functions for every type of Person that exists
- Container classes (linked lists, binary trees, etc.) can be defined to hold a Person, and can hold any subclass of Person
  - Allows limited heterogeneity in container classes



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



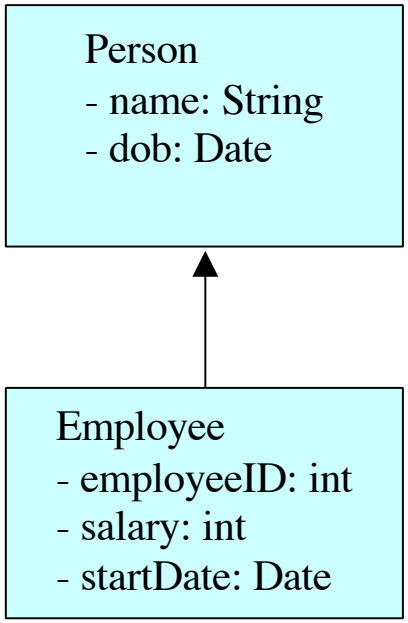
Inheritance in Java

- 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 employeID;
  private int salary;
  private Date startDate;
  [...]
```

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



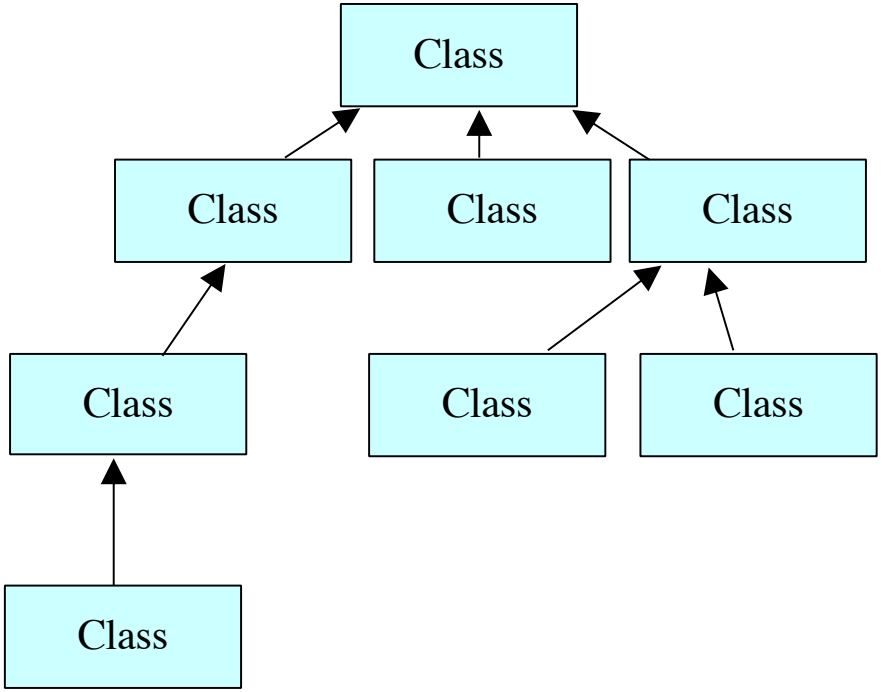
Inheritance Hierarchy

- 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

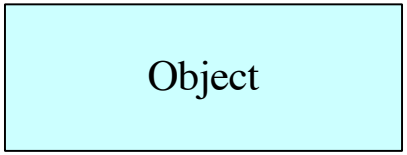
- There is no limit to the number of subclasses a class can have

- 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



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 (default) 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;
  }
}
```

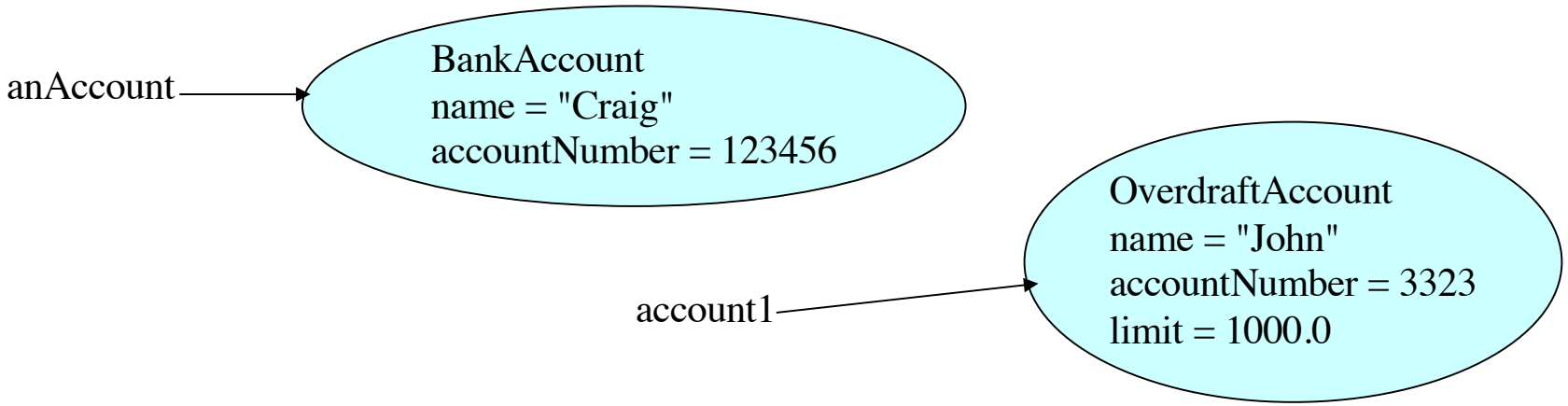
Method overriding - Example

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

  @Override  // This provides a compiler check of signature
  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);
```



Abstract classes / methods

- Abstract methods specified by super class, implemented by all subclasses
- If there is at least one abstract method, then no objects of super class can be created (can still be specified as input parameter to a method, though)
- All abstract methods = interface
- Mix of abstract, not abstract methods allows flexibility in providing shared functionality and specifying required methods subclass must implement

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.

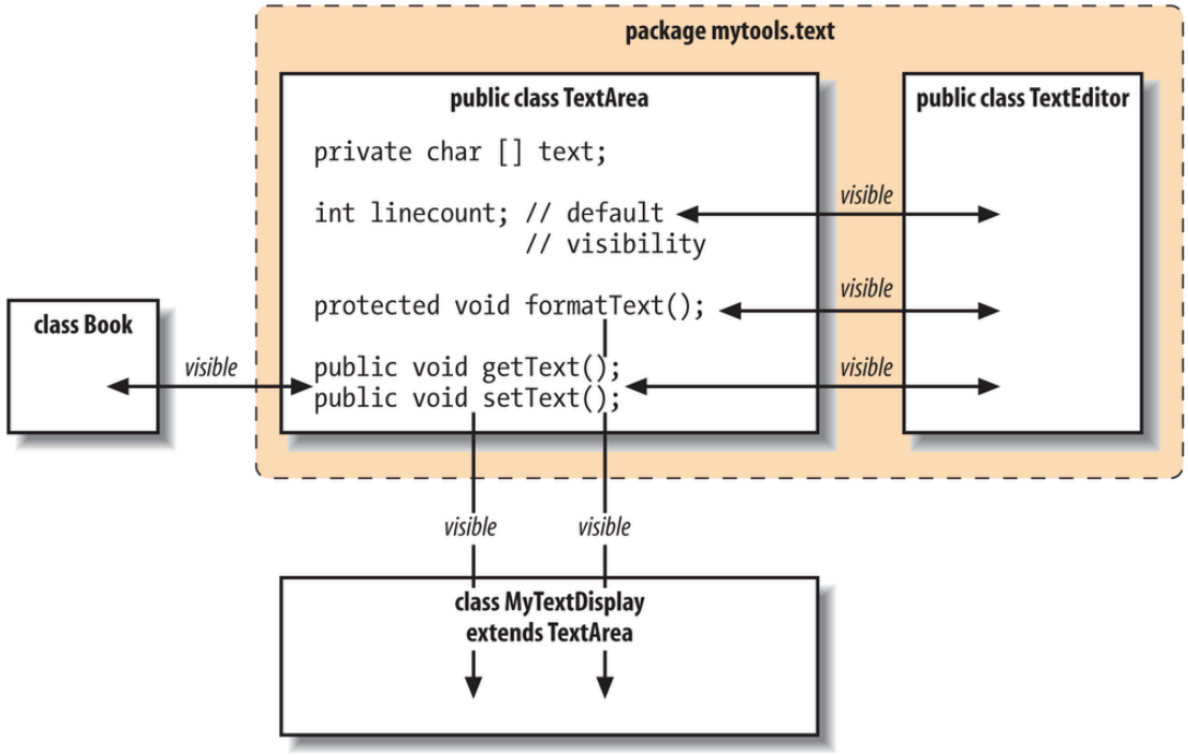
```
public final boolean validatePassword(String username, String Password)
{
  [...]
```

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

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

Permissions

Modifier	Visibility outside class
`private`	None
`no modifier`	Classes in the package
`protected`	Classes in package & all subclasses
`public`	All classes



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?