Introduction to data structures, algorithms, and object-oriented programming
Inheritance &
Class Hierarchies
class Student
{
    private String name;
    private Date dateOfBirth;
    ...
}

- a **Student** has-a **String** for its name
- a **Student** has-a **Date** for its **dateOfBirth**
In UML Terms

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>String name</td>
</tr>
<tr>
<td>Date dateOfBirth</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Terminology for Inheritance

- parent class
- superclass
- base class
- child class
- subclass
- derived class
Person

name
ssn
dateOfBirth
address
...


↑

Student

major
gpa
classList
transcript
advisor
...


Syntax for Inheritance

• class Student extends Person
is-a Relationship

• a Student is-a Person

• Every Student object is also a Person object.

• Class Student inherits data & methods from Class Person.
Inheritance Promotes Software Reuse

- Start with a class which is almost what we want.
- Extend it and add additional capabilities to the subclass.
Probable Issues

• What methods and variables of the parent can the child see?

• What if the child wants to add a method or variable *with the same name* as a method or variable of the parent?
Why Not Modify Existing Code?

• Source code may not be available to us.
• Existing code may be difficult to understand.
• Our changes may break the existing code.
Person

<table>
<thead>
<tr>
<th>String name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person(String)</td>
</tr>
<tr>
<td>String toString()</td>
</tr>
</tbody>
</table>

Student

<table>
<thead>
<tr>
<th>String major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student(String,String)</td>
</tr>
<tr>
<td>String toString()</td>
</tr>
</tbody>
</table>
class Person
{
    protected String name;

    public Person ( String name )
    {
        this.name = name;
    }

    public String toString() { return name; }
}
public Student ( String name, String major )
{
    super(name);
    this.major = major;
}
public String toString()
{
    return name + " (" + major + ")";
}
class Person
{
    private String name;

    public String toString() { return name; }
}
class Student extends Person {
    private String major;

    public String toString() {
        return super.toString() + " (" + major + ")";
    }
}
• Use the parent's constructor to validate parental variables.
Object — The Top of All Hierarchies

- public String toString()
- public boolean equals(Object)
• Don't confuse overriding with overloading.
Single vs Multiple Inheritance

Car    Truck

Minivan
• Java only allows *single* inheritance, so a child class can only have one parent, or one subclass can have only one superclass.

• Interfaces can be used to achieve the effects of multiple inheritance.
class Animal

class Dog extends Animal

class Cat extends Animal

class Human extends Animal

class Mime extends Human

• Each has
  public void speak()
• **Animal:**
  System.out.println("* generic animal noise *");
  
• **Dog:**
  System.out.println("woof");

• **Cat:**
  System.out.println("meow");

• **Human:**
  System.out.println("hello");

• **Mime:**
  System.out.println();
Animal[] animals = new Animal[MAX];
animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Human();
animals[3] = new Dog();
animals[4] = new Mime();
animals[5] = new Cat();
animals[6] = new Animal();
for ( int i = 0 ; i < MAX ; i++ )
{
    animals[i].speak();
}
Motivation for Abstract Classes

• We shouldn't instantiate `Animal` — The class is too generic!

• `Animal` is really a *placeholder* in the hierarchy.

• Let's make `Animal` an *abstract class*. 
abstract class Animal
{
    //----------------------------------------------
    abstract public void speak();
    //----------------------------------------------
}
animals[6] = new Animal();

• Prohibited!

• You can't instantiate an abstract class — it has no instances — it's only a placeholder.
abstract class Person
{
    private String name;
    public Person ( String name ) { this.name = name; }
    public String toString() { return name; }
    abstract public double pay();
}

class Volunteer extends Person {
{
    //-------------------------------------------------------------------------------
    public Volunteer ( String name )
    {
        super(name);
    }
    //-------------------------------------------------------------------------------
    public double pay()
    {
        System.out.println("Thank " + super.toString() + ".");
        return 0.0;
    }
    //-------------------------------------------------------------------------------
}
abstract class Employee extends Person
{
    protected String ssn;
    //------------------------------------------------------------------------
    public Employee ( String name, String ssn )
    {
        super(name);
        this.ssn = ssn;
    }
    //------------------------------------------------------------------------
    public String toString() { return super.toString() + " (" + ssn + ")"; }
    //------------------------------------------------------------------------
}
class Salaried extends Employee {
    private double weeklySalary;

    public Salaried ( String name, String ssn, double weeklySalary ) {
        super(name, ssn);
        this.weeklySalary = weeklySalary;
    }

    public double pay() {
        System.out.printf("Pay $%7.2f to %s\n", weeklySalary, super.toString());
        return weeklySalary;
    }
}
class Hourly extends Employee
{
    protected double hourlyRate;
    protected double hoursWorked;

    public Hourly ( String name, String ssn, double hourlyRate )
    {
        super(name,ssn);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = 0.0;
    }

    public void addHours ( double hours ) { hoursWorked += hours; }

    public double pay()
    {
        double amount = hoursWorked * hourlyRate;
        System.out.printf("Pay $%7.2f to %s.\n",amount, super.toString());
        hoursWorked = 0.0;
        return amount;
    }
}
class UnionMember extends Hourly {
    public UnionMember ( String name, String ssn, double hourlyRate ) {
        super(name,ssn,hourlyRate);
        this.hoursWorked = 0.0;
    }

    public double pay() {
        double amount = hoursWorked * hourlyRate;
        if ( hoursWorked > 40 ) amount += 0.5 * (hoursWorked - 40) * hourlyRate;
        System.out.printf("Pay $%7.2f (includes overtime) to %s.\n",
                amount,
                super.toString());
        hoursWorked = 0.0;
        return amount;
    }
}