Inheritance and Polymorphism Part 01
Inheritance allows Data and Methods to be *inherited / absorbed* from one class into another.

In Java, this occurs between two classes:
- Subclass (Child): The class inheriting from another
- Superclass (Parent): The class that is being inherited

This is great for *extending* the properties and functionality of one class into another:
- The subclass becomes a more specific version of the superclass
- The superclass is a more general version of the subclass
- Creates an “is a” relationship
Inheritance

- Private properties and methods are not inherited from the superclass
- Instance variables from the superclass must be accessed through their accessors
- Instance variables from the superclass must be modified through their mutators

Inheritance Concept

Parent

Child
• The reserved word “extends” is used to inherit from one class into another
• The reserved word “super” is used to call methods from the superclass
  – To call a superclass’ constructor it should use “super()” or “super(<parameters>)”
  – To call any other superclass’ methods user “super.<method id>(<parameters>)”
  – To access properties in the superclass use “super.<accessor>”
  – To modify properties in the superclass use “super.<mutator>”

Inheriting a Class Syntax

```java
public class <<subclass id>> extends <<superclass id>>
{
    //Body of the class
}
```

Example

```java
public class Student extends Person
{
    //Body of the class
}
```
• In Java, a subclass does not inherit constructors from the superclass
• The reserved word “super” should be used to call the superclass’ constructor
  – Generally, it should be the first call in a subclass’ constructor
  – “super()” is the call to the superclass’ default constructor
  – “super(<SC’s params>)” is the call to the superclass’ (SC’s) parameterized constructor
• The subclass’ parameterized constructor should also include the superclass’ properties as parameters

Inheritance Constructor Syntax

```java
public <<subclass id>>(){//Default Constructor
    super();//Call to super class’ default constructor
    ...
}
```

Example

```java
public Student()
{
    super();//Call to Person’s default constructor
    ...
}
```
In Java, a subclass does not inherit constructors from the superclass.

The reserved word “super” should be used to call the superclass’ constructor:
- Generally, it should be the first call in a subclass’ constructor.
- “super()” is the call to the superclass’ default constructor.
- “super(<SC’s params>)” is the call to the superclass’ (SC’s) parameterized constructor.

The subclass’ parameterized constructor should also include the superclass’ properties as parameters.

**Inheritance Constructor Syntax**

```java
public <<subclass id>>(<<SC’s params>>, <<subclass’ params>>) {
    //Call to super class’ default constructor
    super(<SC’s params>);
    ...
}
```

**Example**

```java
public Student(String aN, int anID) {
    super(aN); //Call to Person’s param constructor
    ...
}
```
Overriding a method is when a subclass has the same signature or definition of a method in the superclass.

- Different from Overloading Methods

- A method from the superclass can be called from the subclass by using “super.<method id>”

- This can be useful when overriding the “.equals()” and “.toString()” method

- The reserved word “final” can be used so a method or class CANNOT be overridden or extended

Inheritance “toString()” Syntax

```java
public String toString()
{
    return super.toString() + <<other properties>>
}
```

Example

```
//Student’s toString method
public String toString()
{
    return super.toString() + “ID: ”+this.id;
}
//super.toString() is a call to Person’s toString method
```
Overriding a method is when a subclass has the same *signature* or *definition* of a method in the superclass. Different from Overloading Methods.

A method from the superclass can be called from the subclass by using `super.<<method id>>`.

This can be useful when overriding the `"equals()"` and `"toString()"` method.

The reserved word “final” can be used so a method or class CANNOT be overridden or extended.

### Inheritance ‘equals()’ Syntax

```java
public boolean equals(<<AI>>)
{
    return super.equals(<<AI>>) &&
    <<property checks>>
}
```

### Example

// Student’s equals method
```java
public boolean equals(Student aS)
{
    return aS != null && super.equals(aS) &&
    this.id == aS.getID();
}
```

// super.equals() is a call to Person’s equals // method
Example
Problem: We must keep track of important information about people at a University

Different types include:
- Undergraduate Students
- Graduate Students
- Faculty
- Staff

Undergraduate Information
- Name
- Student ID
- Level

Graduate Information
- Name
- Student ID
- Advisor's Name

Faculty Information
- Name
- Salary
- Courses

Staff Information
- Name
- Salary
- Supervisor

Should be able to:
- Add new people
- Remove people
- View all people in the system

Clear and Easy-to-Use Frontend
| -name: String |
Ugrad

-name: String
-id: int
Ugrad

- name: String
- id: int
- level: int
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-name: String</td>
<td></td>
</tr>
<tr>
<td>-id: int</td>
<td></td>
</tr>
<tr>
<td>-level: int</td>
<td></td>
</tr>
<tr>
<td>+toString(): String</td>
<td></td>
</tr>
<tr>
<td>Ugrad</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>- name: String</td>
<td></td>
</tr>
<tr>
<td>- id: int</td>
<td></td>
</tr>
<tr>
<td>- level: int</td>
<td></td>
</tr>
<tr>
<td>+ toString(): String</td>
<td></td>
</tr>
<tr>
<td>+ equals(Ugrad): boolean</td>
<td></td>
</tr>
</tbody>
</table>
Ugrad
- name: String
  - id: int
  - level: int

+ toString(): String
+ equals(Ugrad): boolean

Grad
- name: String
  - id: int
  - advisor: String

+ toString(): String
+ equals(Grad): boolean
<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Student</strong></td>
<td>- name: String</td>
<td>- toString(): String</td>
</tr>
<tr>
<td></td>
<td>- id: int</td>
<td>+ equals(Student): boolean</td>
</tr>
<tr>
<td><strong>Ugrad</strong></td>
<td>- name: String</td>
<td>- toString(): String</td>
</tr>
<tr>
<td></td>
<td>- id: int</td>
<td>+ equals(Ugrad): boolean</td>
</tr>
<tr>
<td><strong>Grad</strong></td>
<td>- name: String</td>
<td>- toString(): String</td>
</tr>
<tr>
<td></td>
<td>- id: int</td>
<td>+ equals(Grad): boolean</td>
</tr>
<tr>
<td></td>
<td>- advisor: String</td>
<td></td>
</tr>
</tbody>
</table>
Ugrad
- name: String
- id: int
- level: int
+ toString(): String
+ equals(Ugrad): boolean

Grad
- name: String
- id: int
- advisor: String
+ toString(): String
+ equals(Grad): boolean

Student
- name: String
- id: int
+ toString(): String
+ equals(Student): boolean
Student

- name: String
- id: int
+ toString(): String
+ equals(Student): boolean

Ugrad

- name: String
- id: int
- level: int
+ toString(): String
+ equals(Ugrad): boolean

Grad

- name: String
- id: int
- advisor: String
+ toString(): String
+ equals(Grad): boolean

"is a" / extends
Student
- name: String
- id: int
+ toString(): String
+ equals(Student): boolean

Ugrad
- level: int
+ toString(): String
+ equals(Ugrad): boolean

Grad
- advisor: String
+ toString(): String
+ equals(Grad): boolean

"is a" /
extends
Example
• Every class in Java extends from the type “Object”
• Many methods are already assumed to be in each class
  – toString()
  – equals()
• We override the “toString()” and “equals()” methods to define our own specific version
  – The “toString()” method is called every time a class’ id is an argument for “System.out.println()”

Inheritance Concept

• Inheritance Concept
  – Object
  – <<All other Classes>>
• “One becomes many”
• A superclass can be extended or implemented in many different ways
• A change to a superclass is reflected across all subclasses
• Allows substitution of one class for another as long as the class is an extension
  – This is how the “equals()” methods works for different types
• Made possible by dynamic binding aka late binding