ADVANCED OOP: INHERITANCE

OVERVIEW

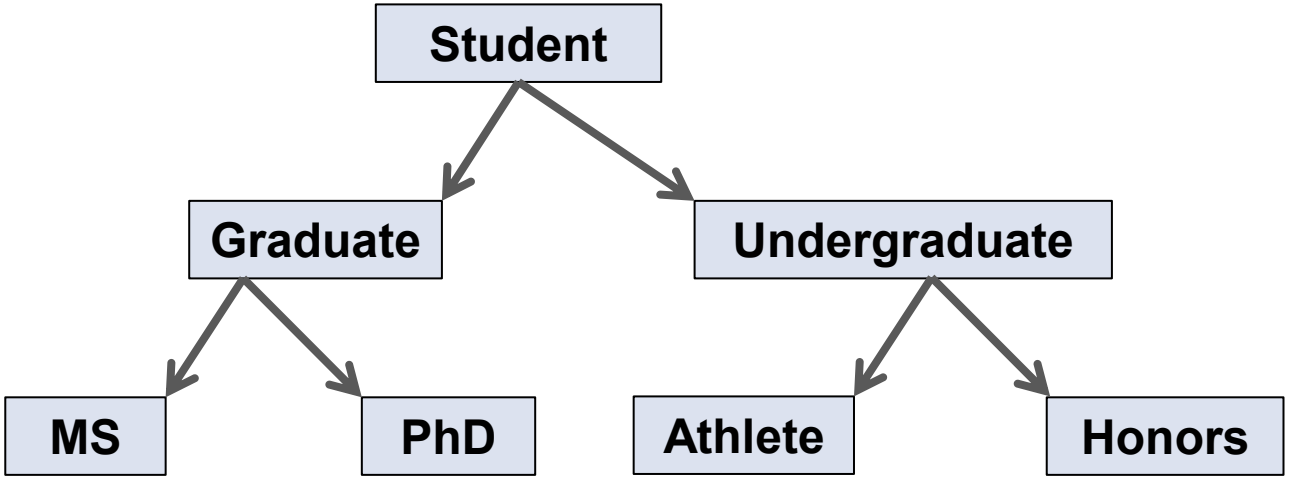
OVERVIEW

- **What is inheritance?**
  - A trait or legacy received from a parent or ancestor
  - Example: money from great uncle
  - Example: brown eyes from mother

- **In Java inheritance is a way to build new classes**
  - We can derive a new class by extending an existing class
  - The new class will inherit all of the fields and methods of the existing class (without having to copy/paste)
  - The derived class is called the subclass (or child)
  - The existing class is called the superclass (or parent)

OVERVIEW

- **Inheritance can be visualized like a family tree**
  - Multiple child classes can be derived from same parent
  - We can have multiple generations of inheritance
  - We are <u>not</u> allowed to inherit from multiple parents



OVERVIEW

- **Why is inheritance important?**
  - It saves development time and reduces code duplication by basing new classes on existing classes
  - This increase reliability by extending debugged classes
  - We can also use polymorphism to process groups of related objects more efficiently in some applications

- **Lesson Overview**
  - Syntax for inheritance
  - Examples of inheritance
  - Summary

ADVANCED OOP: INHERITANCE

PART 1

SYNTAX

SYNTAX

- **We use the keyword extends in the subclass (child) definition to name the superclass (parent)**
  - `public class Employee extends Person`
  - `public class Car extends Vehicle`
  - `public class Truck extends Vehicle`
  - `public class Hybrid extends Car`

- **In Java multiple children are allowed to inherit from one parent, but a child is <u>not</u> allowed to have multiple parents**
  - Multiple inheritance is allowed in some other languages but the syntax and implementation get very ugly

SYNTAX

- **The keyword super is used in two ways when implementing a subclass to access the superclass**
  - We can call the constructor methods of the superclass using super() or super(params)
  - We can call public methods in the superclass using super.method_name(params)
  - We can not access private data fields of the superclass directly, so we need to call superclass getters and setters
  - We can <u>not</u> access private methods of the superclass

SYNTAX

- **By default a subclass can not directly access private data fields of the superclass**
- **We can change this if we have access to the implementation of the superclass**
- **Use the keyword protected instead of private when defining the data fields of the superclass**
  - private String firstName;
  - private String lastName;
  - protected String homeAddress;
  - protected double GPA;

ADVANCED OOP: INHERITANCE

PART 2

EXAMPLES

EXAMPLES

- **Assume we already have a Person class and we want to create an Employee class**
  - The two objects have many fields in common

Person:
First name
Last name
Birth date

Employee:
First name
Last name
Birth date
Hire date
Employee number
Annual salary

EXAMPLES

Definition of Person class:

```
public class Person
{
    private String firstName;
    private String lastName;
    private String birthDate;

    public Person() { ... }
    public getFirstName() { ... }
    public getLastName() { ... }
    ...
    public print() { ... }
}
```

EXAMPLES

Definition of Employee class:

```
public class Employee
{
    private String firstName;
    private String lastName;
    private String birthDate;
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public getFirstName() { ... }
    public getLastName() { ... }
    ...
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

Copy/paste from the Person class

EXAMPLES

Definition of Employee class:

```
public class Employee
{
    private String firstName;
    private String lastName;
    private String birthDate;
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public getFirstName() { ... }
    public getLastName() { ... }
    ...
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

Add new fields and methods to the Employee class

EXAMPLES

Definition of Employee class:

```
public class Employee
{
    private String firstName;
    private String lastName;
    private String birthDate;
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public getFirstName() { ... }
    public getLastName() { ... }
    ...
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

Edit implementation of some Person methods as needed

EXAMPLES

- **Potential problems:**
  - Employee class has duplicate code from Person
  - Code is longer and more difficult to maintain
  - Any changes to Person methods have to be done to Employee class too (double effort)

- **Solution using inheritance:**
  - Extend the Person class to create Employee class
  - Reuse the private variables without redefining them
  - Reuse (or override) public methods of Person

EXAMPLES

Extend Person to define Employee class:

```
public class Employee extends Person
{
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

We create Employee class by extending the Person class

EXAMPLES

Extend Person to define Employee class:

```
public class Employee extends Person
{
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

We only need to include the new fields and methods

EXAMPLES

Extend Person to define Employee class:

```
public class Employee extends Person
{
    private String hireDate;
    private int employeeNumber;
    private double annualSalary;

    public Employee() { ... }
    public setEmployeeNumber(int n) { ... }
    public setEmployeeNumber(int n) { ... }
    public print() { ... }
}
```

We can create Employee methods that override the Person methods

CODE DEMO

Person.java

Employee1.java

Employee2.java

CODE DEMO

Time2.java

MilliTime1.java

MilliTime2.java

SUMMARY

- **Inheritance is an important OOP feature because it saves development time and increases software robustness**
  - We use the keywords “extends” “super” and “protected” when implementing inheritance in Java
  - We add data fields and methods in the subclass to make it more specific than the general purpose superclass
  - We can <u>override</u> methods in the superclass with more methods with the same signature in the subclass
  - The Java class libraries use inheritance extensively (there are dozens of classes derived from Exception)