Inheritance

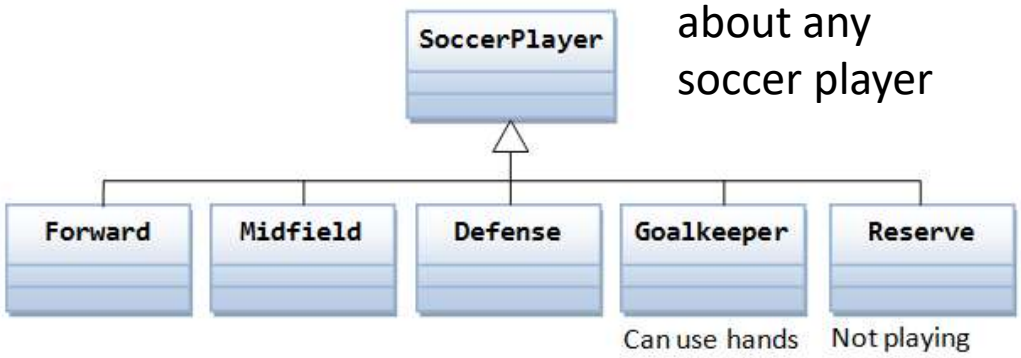
CSCI 2300

Topics Overview

- Organize classes into hierarchy, to avoid code duplication
- Form of ‘generalization’
- Put all common variables and methods into the parent class
- Put specialized variables and methods into subclasses
- Subclass inherits variables and methods of the parent class (and ‘grand-parent’ class)
- Subclass can override methods of the parent class – change the behavior

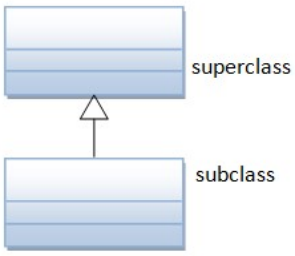
Organize classes into hierarchy, to avoid code duplication

Generalization about any soccer player



Java Syntax and UML (Unified Modeling Language)

```
public class GoalKeeper extends SoccerPlayer{ }
```



```
public class Point2D
{
  private int x;
  private int y;
  public Point2D(int x, int y);
  public int getX();
  public int getY();
}
```

Put all common variables and methods into the parent class

Put specialized variables and methods into subclasses

```
public class Point3D extends Point2D
{
  private int z;
  public Point3D(int x, int y, int z)
  {
    super(x, y);
    this.z = z;
  }
  public int getZ();
}
```

Keyword **super** refers to parent class

Keywords 'super' and 'this'

- **`this`** – keyword that can be used inside a class to refer to the calling object
- **`super`** – keyword that can be used inside a class to refer to the parent class
- **`super(args)`** – calls the constructor of the parent class
  - If used in the constructor of a derived class, must be the first thing that happens

Which of the following calls is invalid?

```
A. Point3D p3 = new Point3D(0,0,0);
B. int x = p3.getX();
C. int y = p3.getY();
D. int z = p3.getZ();
E. None: all are valid
```

```
public class Point2D
{
  private int x;
  private int y;
  public Point2D(int x, int y);
  public int getX();
  public int getY();
}
```

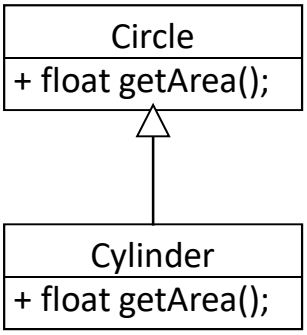
```
public class Point3D extends Point2D
{
  private int z;
  public Point3D(int x, int y, int z);
  public int getZ();
}
```

```
public class Point2D
{
  private int x;
  private int y;
  public Point2D(int x, int y);
  public int getX();
  public int getY();
}
```

Subclass has access to all <u>public</u> and <u>protected</u> variables and methods of the parent class

```
public class Point3D extends Point2D
{
  private int z;
  public Point3D(int x, int y, int z)
  {
    this.x = x;  // COMPILER ERROR
    super(x,y);  // OK
    this.z = z;  // OK
  public int getZ();
}
```

Subclass can **override** (change) the behavior of a parent class




```
public class Cylinder extends Circle
{
  private int height;
  public Cylinder(int r, int h);
  @Override
  public float getArea()
  {
    float circleArea = super.getArea();
    float circ = super.getCircumference();
    return 2*circleArea + circ * height;
  }
}
```

Method overriding

- @Override – annotation that asks compiler to check whether there is such a method in the superclass to be overridden
  - Optional
  - Nice to have
  - Helps if you misspell the method name
  - Has no effect on the execution of the program

Default – no arguments constructor

- If no constructor is defined, a default constructor is automatically generated

```
public ClassName()
{
    super();
}
```

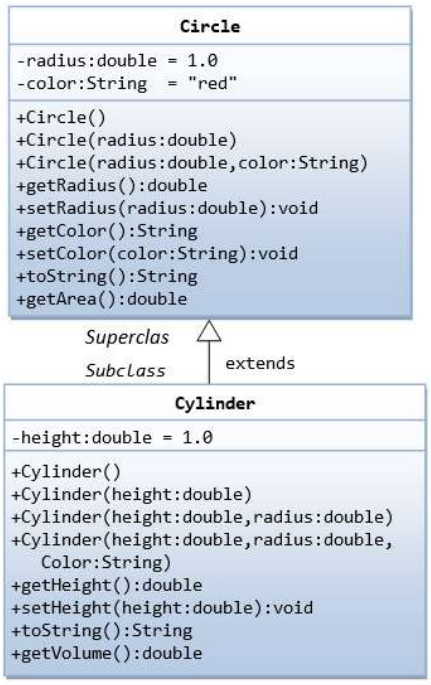
```
public class Document
{
  public Document(String title);
}
```

```
public class Book extends Document
{
  public void setTitle(String t);
  public void setAuthor(String a);
  public String getTitle()
}
```

Consider the instantiation of the Book class below:

```
Book b = new Book();
```

A. This code will work.
B. This code will not work because Book does not have a no-argument constructor.
C. This code will not work because Document does not have a no-argument constructor.
D. This code will not work because Document only has a constructor



Example: Circle and Cylinder

- Reusability principle of OOP
  - Reusing Circle class
- Cylinder inherits methods from the Circle class
- Cylinder overrides methods of the Circle class
- Cylinder adds cylinder specific variable and methods
- Code example on the class site

Lab 4: create lab4 directory in csci2300

- You are designing software for an auto dealership. The dealership sells cars, trucks, and motorcycles. All vehicles are identified by their make, model, and year. Vehicles differ in the following features:
  - Trucks
    - Bed length, number of doors, towing capacity
  - Cars
    - Number of doors, body style: sedan/station wagon/mini-van
  - Motorcycles
    - Type: street, off-road, dual purpose
    - Can have a sidecar
- Design classes to represent the information of your auto dealership. Write a Driver class that creates an object of each class you wrote and prints out attribute information of your objects. Make sure to add `toString()` method to each class to make printing easier.