Inheritance and Polymorphism

Inheritance (Extends)
Overriding methods
IS-A Vs. HAS-A
Polymorphism
Inheritance (semantics)

- We now have two classes that do *essentially* the same thing
  - The fields are exactly the same
  - The methods are nearly exactly the same
- Duplicated code is "a bad thing"
  - Difficult to maintain all versions in same state
  - More work, period
- Objects can *share* functionality by *extending* another object
  - All fields and methods from the parent class (superclass) are automatically copied/available in the child class (subclass).
Inheritance (syntax): extends

- Point is more “basic” than CanvasPoint
  - CanvasPoint is-a Point.
  - CanvasPoint inherits functionality from Point
  - CanvasPoint (subclass) extends Point (superclass)
- The subclass can add new fields/methods
- The subclass can override methods that it inherits!

```java
public class CanvasPoint extends Point {
    public CanvasPoint ( int x, int y ) {
        // our constructor code here
    } // end constructor
    public void setX ( int x ) {
        // our code to check valid x here
    } // end method setX
    public void setY ( int y ) {
        // our code to check valid y here
    } // end method setY
} // end class CanvasPoint
```

```
Point
-x
-y
+Point( x : int, y : int )
+display() : void
+setX(x : int) : void
+setY(y : int) : void
+getX() : int
+GetY() : int
+toString() : String
+equals(obj : Point) : boolean
+clone() : Point
CanvasPoint
+CanvasPoint( x : int, y : int )
+setX(x : int) : void
+setY(y : int) : void
+clone() : CanvasPoint
```
Inheritance (syntax): super

- The subclass inherits all accessible superclass members
  - Private members are not directly accessible
- If you want to use a superclass’s method instead of the subclasses override then you can specify `super`

```java
public class CanvasPoint extends Point {
    public void setX ( int x ) {
        if (x < 0) {x = 0};
        if (x > 800) {x = 800};
        super.setX(x);
    } // end method setX
    // ...
}
```

```java
public class CanvasPoint extends Point {
    public void setX ( int x ) {
        if (x < 0) {x = 0};
        if (x > 800) {x = 800};
        this.x = x; // fails!! x is private
    } // end method setX
    // ...
}
```
Inheritance (syntax): protected

- A subclass inherits all accessible superclass members
- *protected* access modifier: allows full access to subclasses (cool) and all classes in package (horrible!)
- Style hint: Use super to avoid access modifier protected

```java
public class CanvasPoint extends Point {
    public void setX ( int x ) {
        if (x < 0) {x = 0};
        if (x > 800) {x = 800};
        this.x = x;    // ok if x is accessible
    } // end method setX
    ...
} // end class CanvasPoint

public class CanvasPoint extends Point {
    public void setX ( int x ) {
        if (x < 0) {x = 0};
        if (x > 800) {x = 800};
        super.setX(x);
    } // end method setX
    ...
} // end class CanvasPoint
```

Good style!

Horror!

Wright State University, College of Engineering
Dr. T. Doom, Computer Science & Engineering
Inheritance (style): constructors

- **Style**: Always explicitly call the superclass constructor as the very first line of the subclass constructor.

- **Syntax**: There are other valid syntactically correct ways to implement subclass constructors. Avoid them for now!

```java
public class CanvasPoint extends Point {
    CanvasPoint(int x, int y) {
        super(x,y);
        setX(x); // apply CanvasPoint mutators
        setY(y); // to enforce CanvasPoint bounds
    } // end constructor

    // end class CanvasPoint
}
```

---

Wright State University, College of Engineering
Dr. T. Doom, Computer Science & Engineering

CS 241
Computer Programming II
Inheritance trees

- You can extend most classes
  - You can extend subclasses
  - No practical limit to depth
  - Most trees are shallow and wide
- Classes declared `final` cannot be extended (rare!)
- Objects call the “lowest” method from their start in the tree!
- How many instance variables does each class have?
- What methods are available to each class?

ClassroomTeacher `is-a` Teacher
ClassroomTeacher `has-a` `classSize`
Professor `is-a` ClassroomTeacher and `is-a` Teacher

Teacher
- name : String
- institution : String
- area : String
- teach() : Knowledge

CorporateTrainer
- promoteCorp() : BS
- teach() : Knowledge

ClassroomTeacher
- classSize : int
- settleClassDown() : BS

Professor
- research() : Knowledge
- pontificate() : BS
Inheritance (style)

- Look for objects that have common attributes and behaviors:
  - Do they know some of the same things?
  - Do they do some the same things?
- Design a class that represents the common state and behavior
  - This avoids duplicated code and promotes extendibility!
- Decide if a subclass needs behaviors (method implementations) that specific to the subclass type
  - Override as needed! Use `super` to extend methods if appropriate.
- Look for more opportunities to use abstraction by finding two or more subclasses that might need common behavior
  - Make your tree as wide / deep as necessary.
- Write out the final class hierarchy as a UML diagram
  - UML is like the blueprint for object design
  - Is-a is unidirectional! Know the direction!
  - Superclasses cannot inherit anything from subclasses!
Inheritance (reference summary)

- A subclass extends as superclass
- A subclass inherits all public instance variables and methods of the superclass, but does not inherit the private instance variables and methods of the superclass
- Inherited methods can be overridden; instance variables cannot be overridden
- Use the IS-A test to verify your inheritance hierarchy. If X extends Y then X IS-A Y must make sense
- The IS-A relationship only works in one direction. A penguin is-a bird, but not all birds are penguins.
- When a method is overridden in a subclass, and that method is invoked on an instance of the subclass, the overridden version of the method is called. (The lowest one wins.)
- Inheritance is transitive. If class B extends class A, and C extends B, class B IS-A class A and class C IS-A class B, and class C also IS-A class A.
- You get rid of a lot of duplicated code by using inheritance. If you need to change the shared behavior, you just have to update in the superclass. All classes that extend it will automatically use the new version!
Polymorphism

- Inheritance buys us two major things
  - (1) You avoid duplicate code
  - (2) You define a *common protocol* for a set of classes related through inheritance.
- When you define an inheritable method in a superclass, all objects that extend it *must* implement that method.
  - They may use the method as-is in the superclass
  - They may override it with their own implementation
  - But they have it
- This establishes a contract and allows *polymorphism* (many shapes)
  - An instance of any object is considered to match type with all of its superclasses.
  - Any object that passes the IS-A test with any class can be stored/passed/returned as a reference of that type!
Polymorphism in action

... Teacher smith = new Teacher();
Teacher doom = new Professor();
...
if (doom.institution.equals(smith.institution)) {
    ...
}
...

Teacher[] teacherList;
teacherList = new Teacher[10];
teacherList[0] = new Professor();
teacherList[1] = new CorporateTrainer();
...
class Room {
    public void activity(Teacher theTeacher) {
        theTeacher.teach();
    }
} // end class activity

class Main {
    public static void main(String args) {
        Room r = new Room();
        CorporateTrainer c = new CorporateTrainer();
        Professor p = new Professor();
        r.activity(c);
        r.activity(p);
    } // end method main
} // end class Main

CorporateTrainer’s overridden teach() runs
Professor’s inherited teach() runs
Polymorphism (summary)

- If you write code using polymorphic arguments (where to declare method parameters as a superclass type) then you can pass in any subclass type at runtime
- This means that after you write/test your code, you (or anyone else) can add *new* subclass types to the program and that your existing methods will still work!
- Similarly, it means that you can extend existing classes (such as library classes) and that all of the library code will work on/with your new class!

- The methods ARE the contract
  - The method name, return value, and arguments list (number, type, order) of the specified supertype is used during compiletime
  - The method name, return value, and arguments list (number, type, order) of the object type is used during runtime
The Object class

- What makes an object an object?
- All classes either:
  - *explicitly* extend another class
  - *implicitly* extend Object
- Class Object is the mother of all classes
- Class Object is the superclass of all objects
- Class Object has the default constructor and other default methods
- The existence of this common superclass allows the creation of routines that (using polymorphism) work on all classes!
  - Consider that you can put objects that *you* define in an ArrayList!
How can you determine if a polymorphic object is an instance of a particular class?

```java
ArrayList<Teacher> teacherList = new ArrayList<Teacher>();
CorporateTrainer c = new CorporateTrainer();
teacherList.add(c);
Professor p = new Professor();
teacherList.add(p);
...
int numProfessors = 0;
int numCorporateTrainers = 0;
for (Teacher teacher : teacherList) {
    if (teacher instanceof Professor) {
        numProfessors++
    }
    if (teacher instanceof CorporateTrainer) {
        numCorporateTrainers++;
    }
}
...
Some *final* thoughts

- **final variables:** value can’t be changed.
  
  ```java
  ...  
  final public double PI = 3.14;  
  ...  
  ```

- **final methods:** method can’t be overridden
  
  ```java
  ...  
  final public String toString() {
  ...  
  }  
  ...  
  ```

- **final class:** class can’t be extended
  
  ```java
  final public class SmartestPersonInTheWorld {
  ...  
  }  
  ```