Inheritance and Polymorphism Part 02
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 Concept**

![Inheritance Diagram](image)
“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*
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0;i<people.length;i++)
    System.out.println(people[i]);
```
• “Many Forms”
• Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

**Polymorphism Example**

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0;i<people.length;i++)
    System.out.println(people[i]);
```

**Console**
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2",4);
people[2] = new Ugrad("asdf3",5,2);

for(int i=0;i<people.length;i++)
    System.out.println(people[i]);
```
“Many Forms”
Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example
```
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2",4);
people[2] = new Ugrad("asdf3",5,2);
for(int i=0;i<people.length;i++)
  System.out.println(people[i]);
```
“Many Forms”

Actions / Functionality (methods) can be implemented in many different ways:
  - equals method
  - toString method

Allows changes in subclass methods to be applied to the inherited superclass

---

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0; i<people.length; i++)
    System.out.println(people[i]);
```

---

Console
• “Many Forms”
• Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

```
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2",4);
people[2] = new Ugrad("asdf3",5,2);

for(int i=0;i<people.length;i++)
System.out.println(people[i]);
```
• “Many Forms”
• Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2",4);
people[2] = new Ugrad("asdf3",5,2);

for(int i=0;i<people.length;i++)
    System.out.println(people[i]);
```
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
public String toString()
{
    return "Name: " + this.name;
}
```
“Many Forms”

Actions / Functionality (methods) can be implemented in many different ways
   – equals method
   – toString method

- Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0;i<people.length;i++)
    System.out.println(people[i]);
```

Console

Name: asdf
“Many Forms”

Actions / Functionality (methods) can be implemented in many different ways
- equals method
- toString method

- Allows changes in subclass methods to be applied to the inherited superclass
• “Many Forms”
• Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
public String toString()
{
    return "Name: " + this.name;
}
```

Console

Name: asdf
• “Many Forms”
• Actions / Functionality (methods) can be \textit{implemented} in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

\textbf{Polymorphism Example}

```java
public String toString()
{
    return super.toString()+" ID: "+this.id;
}
```

\textbf{Console}

Name: asdf
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0; i<people.length; i++)
    System.out.println(people[i]);
```

Console

Name: asdf
Name: asdf2 ID: 4
• “Many Forms”
• Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

```
public String toString()
{
    return super.toString() + " Level: "+this.level;
}
```

**Console**
Name: asdf
Name: asdf2 ID: 4
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
public String toString()
{
    return super.toString()+" ID: "+this.id;
}
```

Console
Name: asdf
Name: asdf2 ID: 4
Polymorphism

• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
public String toString()
{
    return "Name: " + this.name;
}
```

Console
Name: asdf
Name: asdf2 ID: 4
“Many Forms”

Actions / Functionality (methods) can be *implemented* in many different ways
- equals method
- toString method

- Allows changes in subclass methods to be applied to the inherited superclass

**Polymorphism Example**

```java
public String toString() {
    return super.toString()+" ID: "+this.id;
}
```

**Console**

Name: asdf
Name: asdf2 ID: 4
“Many Forms”
Actions / Functionality (methods) can be *implemented* in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

```
public String toString()
{
    return super.toString() + " Level: " + this.level;
}
```
• “Many Forms”
• Actions / Functionality (methods) can be implemented in many different ways
  – equals method
  – toString method
• Allows changes in subclass methods to be applied to the inherited superclass

Polymorphism Example

```java
Person[] people = new Person[3];
people[0] = new Person("asdf");
people[1] = new Student("asdf2", 4);
people[2] = new Ugrad("asdf3", 5, 2);

for(int i=0; i<people.length; i++)
    System.out.println(people[i]);
```

Console

Name: asdf
Name: asdf2 ID: 4
Name: asdf3 ID: 5 Level: 2
• Similar to a Class
  – Creates a Type
  – The identifier of an interface MUST match the filename
• Defines the functionality (methods) a class MUST implement
• Creates a non-constructible Type
  – Can only construct Classes that implement an interface
  – Classes that implement an interface can be assigned to variables of that interface type
• Only Contains method signatures
  – No method body or functionality
  – No instance variables
• “Blueprints for Classes”

Creating an Interface Syntax

```java
public interface <<id>> {
    <<method signatures>>;
}
```

Example

```java
public interface Shape {
    public void setHSpace(int aH);
    public int getHSpace();
    public void drawShape();
    public void drawShapeAt(int lineNumber);
}
```
Reserved word “implements” is used between a class and an interface.

If a method is not defined in a class that implements an interface then the class will have a syntax error.

Useful for when the functionality of a class can be done in a variety of ways.

### Class using an Interface Syntax

```java
public class <<class id>> implements <<interface id>>
{
    <<methods from the interface must be defined in this class>>
}
```

### Example

```java
public class BasicShape implements Shape
{
    //Methods setHSpace, getHSpace, drawShape,
    //and drawShapeAt must be defined in here
}
```
• Declaring a variable of an interface-type is the same as declaring a variable of a class-type
  – Type followed by an identifier
  – Identifiers have the same rules as every other variable identifier

• Cannot construct an instance (object) of an interface
  – Interfaces are non-constructible types

• Only Classes that implement the interface can be constructed an assigned

---

**Using an Interface as a Type Syntax**

```java
//Declaring a variable using the interface as a type
<<interface id>> <id>>;  
//Creating an instance of class that uses the interface
<id>> = new <<Class Constructor>>;
```

**Example**

```java
//Correct
Shape s = new BasicShape();
//Incorrect, because interfaces cannot be constructed
Shape s2 = new Shape();  //Syntax error here
```
• Problem: We must create a program that can draw a variety of shapes in the console
• Draw Shapes in the console at set locations
  – Horizontal Spacing
  – Vertical Spacing
• Some Shapes mentioned were:
  – Rectangle
  – Triangle
  – Maybe more?

• Shapes could be drawn in a variety of ways
  – Filled
  – Hollow
  – Upside Down Triangle
  – Checkered Rectangle
  – Horizontal Striped Rectangle
  – Vertical Striped Rectangle
  – Etc.
Shape <<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
Shape

<<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Shape
<<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

“implements”
Shape

<<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void
Shape <<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
Shape <<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>

+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
Shape <<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>

+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

"extends"
Shape <<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>

+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
Shape <<interface>>

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape

- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>

+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
Shape <<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void
Shape <<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void

"implements"

"extends"
Shape
<<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int

+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle
<<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

BasicRectangle
- width: int
- height: int

+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void
Shape <<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)

BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void

HollowRectangle
+ drawShape(): void

CheckeredRectangle
+ drawShape(): void
Shape

- setHSpace(int): void
- getHSpace(): int
- drawShape(): void
- drawShapeAt(int): void

BasicShape

- hSpace: int
- setHSpace(int): void
- getHSpace(): int
- drawShape(): void
- drawShapeAt(int): void

Triangle

- getHeight(): int
- setHeight(int)

BasicTriangle

- height: int
- getHeight(): int
- setHeight(int)
+ drawShape(): void

Rectangle

- getWidth(): int
- setWidth(int): void
- getHeight(): int
- setHeight(int)
+ getHSpace(): int
+ setHSpace(int): void
+ drawShape(): void
+ drawShapeAt(int): void

HollowRectangle

+ drawShape(): void

CheckeredRectangle

+ drawShape(): void
Shape <<interface>>
- setHSpace(int): void
- getHSpace(): int
- drawShape(): void
- drawShapeAt(int): void

BasicShape
- hSpace: int
- setHSpace(int): void
- getHSpace(): int
- drawShape(): void
- drawShapeAt(int): void
- skipHSpaces(int): void

Triangle <<interface>>
- getHeight(): int
- setHeight(int)

BasicTriangle
- height: int
- getHeight(): int
- setHeight(int)
- drawShape(): void

HollowTriangle
- getHeight(): int
- setHeight(int)
- drawShape(): void

UpsideDownTriangle
- getHeight(): int
- setHeight(int)
- drawShape(): void

Rectangle <<interface>>
- getWidth(): int
- setWidth(int): void
- getHeight(): int
- setHeight(int)
- drawShape(): void
- drawShapeAt(int): void

BasicRectangle
- width: int
- height: int
- getWidth(): int
- setWidth(int): void
- getHeight(): int
- setHeight(int)
- drawShape(): void

HollowRectangle
- drawShape(): void

CheckeredRectangle
- drawShape(): void
Shape <<interface>>
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void

BasicShape
- hSpace: int
+ setHSpace(int): void
+ getHSpace(): int
+ drawShape(): void
+ drawShapeAt(int): void
+ skipHSpaces(int): void

Rectangle <<interface>>
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void

BasicRectangle
- width: int
- height: int
+ getWidth(): int
+ setWidth(int): void
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void

HollowRectangle
+ drawShape(): void

CheckeredRectangle
+ drawShape(): void

Triangle <<interface>>
+ getHeight(): int
+ setHeight(int)

BasicTriangle
- height: int
+ getHeight(): int
+ setHeight(int)
+ drawShape(): void

UpsideDownTriangle
+ drawShape(): void

HollowTriangle
+ drawShape(): void

Tester
+ main(String[]): void
Keep in mind
- Classes *extends* Classes
- Interfaces *extends* Interfaces
- Classes *implements* Interfaces

In Java, classes can implement several interfaces but only extend one other class
- Extends first followed by Implements
- Each interface that is implemented is separated by a comma

Polymorphism allows software to be very *extensible*