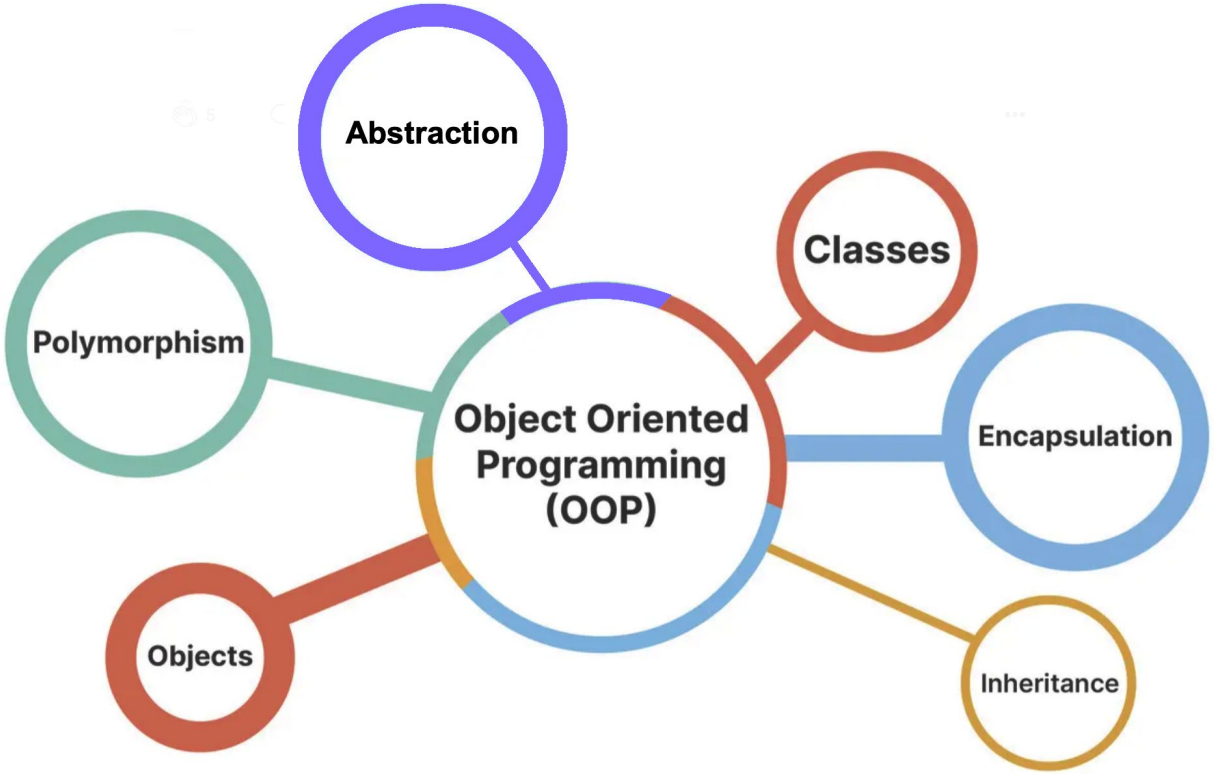
COSC 121: Computer Programming II



Class Relationships You Should Be Familiar With

- Dependency ("uses")
  - A class uses another class
    - Ex: The Dog class uses the Scanner class
  - An object of one class uses another object of the same class
    - Ex: A Dog object shares snacks with another Dog

Class Relationships You Should Be Familiar With

- Dependency ("uses")
  - A class uses another class
    - Ex: The Dog class uses the Scanner class
  - An object of one class uses another object of the same class
    - Ex: A Dog object shares snacks with another Dog
- Aggregation ("has-a")
  - A class has objects of another class
    - Ex: A Library object has many Book objects

Today's Key Concepts

- Inheritance models a new IS-A relationship
- Implementation techniques:
  - Relationship created via extends
  - New visibility modifier: protected
  - Reference to parent object called super
  - The final modifier
  - Method overriding
- Adherence to encapsulation principles
- The Object class
- A class cannot inherit from more than one class

Why Use Inheritance?

- Software reusability
  - Saves time and effort by inheriting methods and attributes from parent class
  - Reduces chances of bugs
- Conceptual organization
  - Clear class hierarchy and code structure
  - Can be extended to add new methods and attributes to child class
  - Can modify (override) inherited method definitions inside child class
- Alternative? Copy & paste code to multiple parts of the program

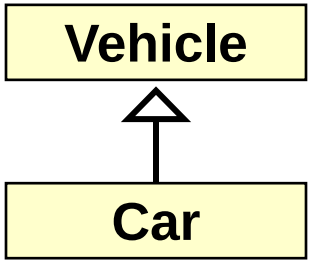
Inheritance ("is-a")

- Inheritance is a mechanism for extending and enhancing existing classes
  - In real life, you inherit some of the properties from your parents, but you also have unique properties specific to you
  - In Java, a class that extends another class inherits its properties (methods, instance variables) and can also override and define properties of its own
- extends is the keyword used to define this relationship
- Syntax: class ChildClass extends ParentClass

Diagrammatically

- Use a box to represent a class
- Use an upward arrow to point to the parent class

parent class/superclass



child class/subclass

A car IS-A vehicle

- A child class inherits from a parent class
- A child class is derived from a parent class

Implementation

- Use a reserved word extends to indicate the relationship
- Template:

```
public class Child extends Parent
{
    // class contents
}
```

- Example:

```
public class Car extends Vehicle
{
    // class contents
}
```

iClicker Question

What are the IS-A relationships defined by the following code?

```
public class Animal { … }
public class Mammal extends Animal { … }
public class Reptile extends Animal { … }
public class Dog extends Mammal { … }
```

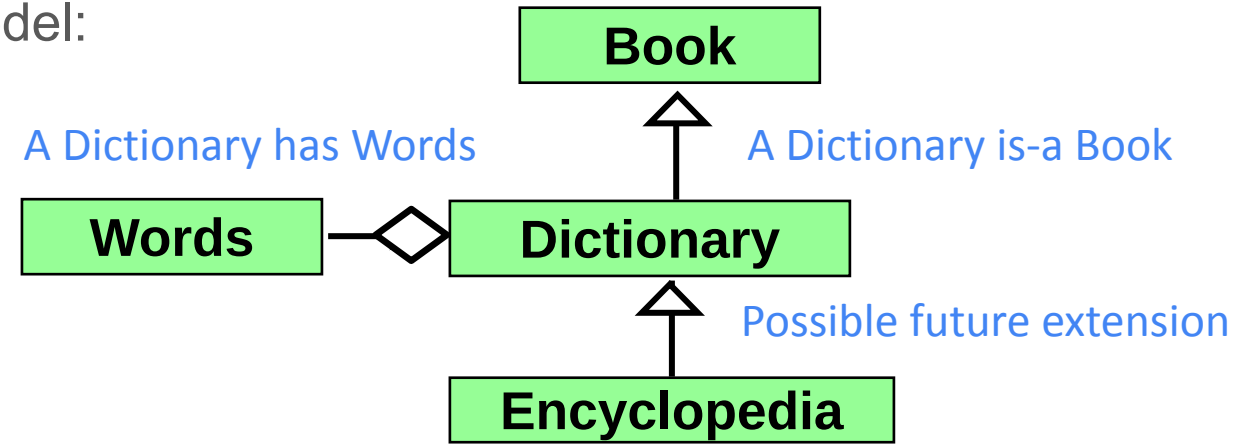
A. Animal is-a Mammal, Animal is-a Reptile, Mammal is-a Dog
B. Animal is-a Animal, Mammal is-a Animal, Reptile is-a Animal, Dog is-a Mammal
C. Mammal is-a Animal, Reptile is-a Animal, Dog is-a Mammal
D. Mammal is-a Animal, Reptile is-a Animal, Dog is-a Mammal, Dog is-a Animal

Longer Example

- Client says:
- "I want a software program that lets me look up word definitions easily. After that, I might also want to extend the program to give me more complicated entries with pictures, like an encyclopedia."
- What classes do we need to model?
- How are they related?

Longer Example

- Client says:
- "I want a software program that lets me look up word definitions easily. After that, I might also want to extend the program to give me more complicated entries with pictures, like an encyclopedia."
- Conceptual model:



A Very Simple Book Class

```
public class SimpleBook
{
  protected int pages;

  public SimpleBook( int maxPages )
  {
    pages = maxPages;
  }

  public void setPages( int numPages ) { pages = numPages; }
  public int getPages()                { return pages; }
}
```

A Very Simple Book Class

```
public class SimpleBook
{
  protected int pages;

  public SimpleBook( int maxPages )
  {
    pages = maxPages;
  }

  public void setPages( int numPages ) { pages = numPages; }
  public int getPages()                { return pages; }
}
```

what is this?
new visibility modifier!

A Very Simple Book Class

```
public class SimpleBook
{
  protected int pages;

  public SimpleBook( int maxPages )
  {
    pages = maxPages;
  }

  public void setPages( int numPages ) { pages = numPages; }
  public int getPages()                { return pages; }
}
```

what is this?
new visibility modifier:
only visible to derived classes

An Initial Dictionary Subclass

```
public class Dictionary extends SimpleBook
{
  private int numDefs;

  public Dictionary( int maxPages, int maxEntries )
  {
    super( maxPages );
    numDefs = maxEntries;
  }

  public void setNumDefs( int newNumDefs ) { numDefs = newNumDefs; }
  public int getNumDefs()                  { return numDefs; }
}
```

An Initial Dictionary Subclass

```
public class Dictionary extends SimpleBook
{
  private int numDefs;

  public Dictionary( int maxPages, int maxEntries )
  {
    super( maxPages );
    numDefs = maxEntries;
  }

  public void setNumDefs( int newNumDefs ) { numDefs = newNumDefs; }
  public int getNumDefs()                  { return numDefs; }
}
```

what is this?

new keyword!

An Initial Dictionary Subclass

```
public class Dictionary extends SimpleBook
{
  private int numDefs;

  public Dictionary( int maxPages, int maxEntries )
  {
    super( maxPages );
    numDefs = maxEntries;
  }

  public void setNumDefs( int newNumDefs ) { numDefs = newNumDefs; }
  public int getNumDefs()                  { return numDefs; }
}
```

what is this?

new keyword:

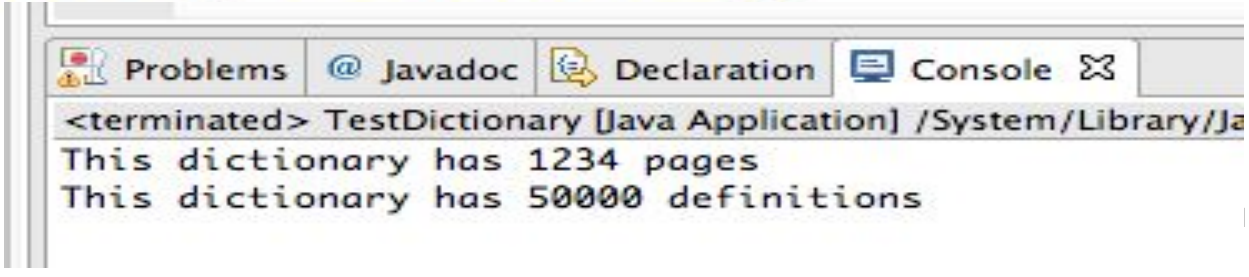
calls constructor in super class

A Test Class

```
public class TestDictionary
{
  public static void main( String[] args )
  {
    Dictionary webster = new Dictionary( 1234, 50000 );
    System.out.println( "This dictionary has "
    + webster.getPages() + " pages" );
    System.out.println( "This dictionary has "
    + webster.getNumDefs() + " definitions" );
  }
}
```

A Test Class

```
public class TestDictionary
{
  public static void main( String[] args )
  {
    Dictionary webster = new Dictionary( 1234, 50000 );
    System.out.println( "This dictionary has "
    + webster.getPages() + " pages" );
    System.out.println( "This dictionary has "
    + webster.getNumDefs() + " definitions" );
  }
}
```



What is Inherited?

- All attributes from the parent class

- All methods from the parent class
  - Exception:

What is Inherited?

- All attributes from the parent class
  - Even private ones
  - How to access them? (See Section 9.4)
  - Note: private variables from parent class can only be accessed using methods defined by parent class
- All methods from the parent class
  - Exception:

What is Inherited?

- All attributes from the parent class
  - Even private ones
  - How to access them? (See Section 9.4)
  - Note: private variables from parent class can only be accessed using methods defined by parent class
- All methods from the parent class
  - Exception: constructors are not inherited *(why not?)*

What is Inherited?

- All attributes from the parent class
  - Even private ones
  - How to access them? (See Section 9.4)
  - Note: private variables from parent class can only be accessed using methods defined by parent class
- All methods from the parent class
  - Exception: constructors are not inherited *(why not?)*
  - Inherited methods can be adapted/overridden to accommodate design of child class

Adding the Word Class

```
public class Word
{
  private String vocab;
  private String pronounciation;
  private String definition;

  public Word( String entry, String sound, String explain )
  {
    vocab = entry;
    pronounciation = sound;
    definition = explain;
  }

  // various accessors and mutators
}
```

Incorporating Word into Dictionary

- How to keep track of Word objects?

Incorporating Word into Dictionary

- How to keep track of Word objects?
  - Each Dictionary should have a list of Word objects
    → new instance variable entries
    - What data structure to use?

Incorporating Word into Dictionary

- How to keep track of Word objects?
  - Each Dictionary should have a list of Word objects
    → new instance variable entries
    - What data structure to use?
    - Where should a new entry entry be stored?
      → new instance variable currWord

Incorporating Word into Dictionary

- How to keep track of Word objects?
  - Each Dictionary should have a list of Word objects
    → new instance variable entries
    - What data structure to use?
    - Where should a new entry entry be stored?
      → new instance variable currWord
  - Words need to be added to the Dictionary
    → new method for addEntry()
    - What input parameters should addEntry() take?
    - What return type should addEntry() have?

Incorporating Word into Dictionary

- How to keep track of Word objects?
  - Each Dictionary should have a list of Word objects
    → new instance variable entries
    - What data structure to use?
    - Where should a new entry entry be stored?
      → new instance variable currWord
  - Words need to be added to the Dictionary
    → new method for addEntry()
    - What input parameters should addEntry() take?
    - What return type should addEntry() have?
- How to test your new changes in TestDictionary?

Sample Solution

```
public class Dictionary extends SimpleBook
{
  private int    numDefs;
  private Word[] entries;
  private int    currWord;

  public Dictionary( int maxPages, int maxEntries )
  {
    super( maxPages );
    numDefs = maxEntries;
    entries = new Word[numDefs];
    currWord = 0;
  }
```

new instance vars

Sample Solution (cont.)

```
public void addEntry( String entry, String pron, String defn )
{
  Word vocab = new Word( entry, pron, defn );
  if( currWord < numDefs )
  {
    entries[ currWord ] = vocab;
    currWord++;
  }
}
```

Sample Solution (cont.)

```
public void addEntry( String entry, String pron, String defn )
{
  Word vocab = new Word( entry, pron, defn );
  if( currWord < numDefs )
  {
    entries[ currWord ] = vocab;
    currWord++;
  }
}
```

// second option for flexibility

```
public void addEntry( Word vocab )
{
  if( currWord < numDefs )
  {
    entries[ currWord ] = vocab;
    currWord++;
  }
}
```

Sample Solution (cont.)

```
  public int  getNumEntries()                { return currWord; }
  public void setNumDefs( int newNumDefs ) { numDefs = newNumDefs; }
  public int  getNumDefs()                   { return numDefs; }
}
```

// adds basic accessors and mutators

// can also define own toString()

- summarizes the contents of the object in a readable way
- default one prints reference address

Testing Sample Solution

- Call the methods you created
- Check outputs before and after

```
public class TestDictionary
{
  public static void main( String[] args )
  {
    Dictionary webster = new Dictionary( 1234, 50000 );
    System.out.println( "This has " + webster.getPages() + " pages" );
    System.out.println( "This has " + webster.getNumDefs() + " definitions" );
    System.out.println( "This has " + webster.getNumEntries() + " entries" );

    webster.addEntry( "key", "ki", "tool used to unlock something" );
    System.out.println( "This has " + webster.getNumEntries() + " entries" );
  }
}
```

iClicker Question

Suppose the purple box has input `y`, what is the output of this program?

A. 0
B. 1
C. 2
D. 3
E. Error

```
class A {
   public int x;
   public void display() {
      System.out.println(x);
   }
}
class B extends A {
   public int y;
   public void display() {
      System.out.println(    );
   }
}

class Test {
   public static void main(String args[]){
      B b = new B();
      b.x = 1;
      b.y = 2;
      b.display();
   }
}
```

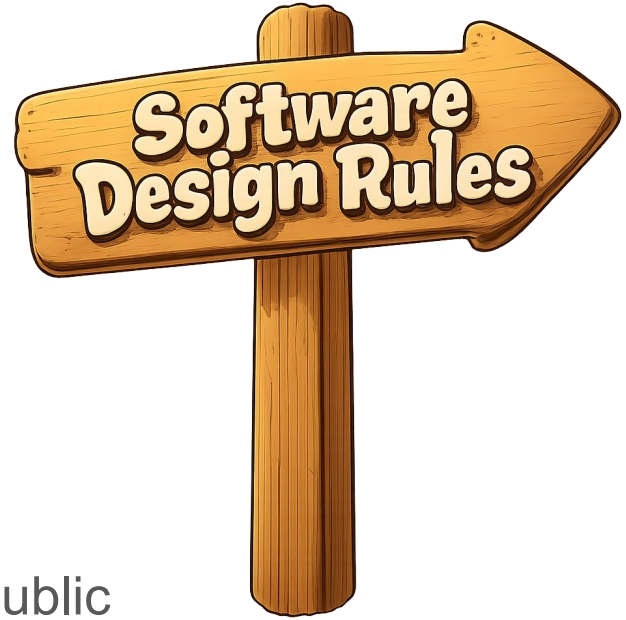
iClicker Question

Suppose the purple box has input `x`, what is the output of this program?

A. 0
B. 1
C. 2
D. 3
E. Error

```
class A {
   public int x;
   public void display() {
      System.out.println(x);
   }
}
class B extends A {
   public int y;
   public void display() {
      System.out.println(    );
   }
}

class Test {
   public static void main(String args[]){
      B b = new B();
      b.x = 1;
      b.y = 2;
      b.display();
   }
}
```

Visibility Modifiers Revisited



- Previously, you saw:
  - No visibility modifier (called “default”)
  - public
  - private
- **Recall encapsulation rules:**
  - Don’t ever leave anything default
  - Unless there’s a reason, classes are public
  - All class attributes are private; access and changes must be done via accessors and mutators
  - Only methods that are to be called by other classes should be public, all other methods (“helpers” within the class) are private

New Visibility Modifier: protected

- Allows a child class to access an attribute or method from the parent class
  - Like granting special access to child classes
  - Trusted classes can see more of the parent class
  - Unrelated classes won't be able to see the info

- Note: protected info is also visible to any class in the same package (not part of this course)

iClicker Question

From the **least** amount of visibility to the **most**, which is the correct order for the visibility modifiers?

A. public (least), protected, private
B. protected (least), private, public
C. private (least), protected, public
D. private = protected (both are the same), then public

Adding to the Encapsulation Rules

- Recall: All class attributes are private; access and changes must be done via accessors and mutators
  - Avoid protected attributes – generally considered as bad practice
- Recommended programming pattern:
  - Prefer private by default
  - Increase visibility only when design calls for it
  - Use private fields with protected getter and setter methods to maintain control

Constructors Revisited: super()

- Constructors are not inherited
  - Even though they have public visibility
  - Constructors are not officially members of the class
  - If they were inherited to the child class as the constructors of the child, the names would be wrong!

Do we need to access the parent class's constructor?

Constructors Revisited: super()

- Constructors are not inherited
  - Even though they have public visibility
  - Constructors are not officially members of the class
  - If they were inherited to the child class as the constructors of the child, the names would be wrong!
- Recall purpose of constructors: to set up attributes
- In many cases, we still want to reuse the parent class's setup
  - Solution: call super() as if you were calling the parent constructor directly and pass in the same input as you would

Calling Parent Constructor Implicitly

- If no constructor is called in child class, Java implicitly calls the the super constructor
- These two are equivalent:

```
class A{
  public A(){
    System.out.print(1);
  }
}

class B extends A{
  public B(){
    System.out.print(2);
  }
}
```

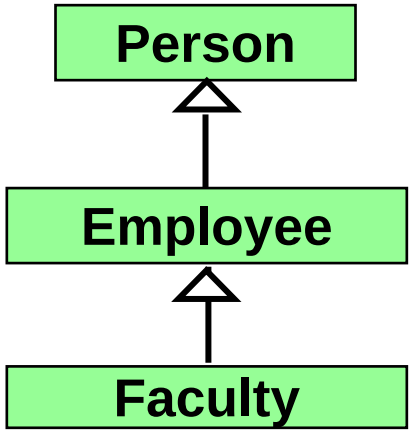
```
class A{
  public A(){
    System.out.print(1);
  }
}

class B extends A{
  public B(){
    super();
    System.out.print(2);
  }
}
```

Output of
B b = new B();
is 12

Constructor Chaining

- When a chain of inheritance relationships are involved, all the parent classes' constructors along the chain are invoked
- Ex:

```
class Person {
  public Person() {System.out.print(1);}
}
class Employee extends Person {
  public Employee() {
    this(2);
    System.out.print(3);
  }
  public Employee(int n) {System.out.print(n);}
}
class Faculty extends Employee {
  public Faculty() {System.out.print(4);}
}
```



Constructor Chaining

- When a chain of inheritance relationships are involved, all the parent classes' constructors along the chain are invoked
- Ex:

```
class Person {
  public Person() {System.out.print(1);}
}
class Employee extends Person {
  public Employee() {
    this(2);
    System.out.print(3);
  }
  public Employee(int n) {System.out.print(n);}
}
class Faculty extends Employee {
  public Faculty() {System.out.print(4);}
}
```

- Test class:

```
public static void main(String[] args) {
   Faculty f = new Faculty();
}
```

// output?

Constructor Chaining

- When a chain of inheritance relationships are involved, all the parent classes' constructors along the chain are invoked
- Ex:

```
class Person {
  public Person() {System.out.print(1);}
}
class Employee extends Person {
  public Employee() {
    this(2);
    System.out.print(3);
  }
  public Employee(int n) {System.out.print(n);}
}
class Faculty extends Employee {
  public Faculty() {System.out.print(4);}
}
```

- Test class:

```
public static void main(String[] args) {
   Faculty f = new Faculty();
}
```

// output: 1234

More on super

- Can use super to call other methods and attributes in the parent class
- Examples in Dictionary.java:

```
super.setPages( 5000 );
super.pages = 2;
// why no brackets?
```

More on super

- Can use super to call other methods and attributes in the parent class
- Examples in Dictionary.java:

  ```
  super.setPages( 5000 );
  super.pages = 2;
  // why no brackets?
  ```
- super is a reference variable to the immediate parent object
- Be careful not to break encapsulation rules!
  - Use accessors and mutators when possible