1 **Objected-Oriented Programming: Inheritance**

CST242

2 **OOP**

- Object-Oriented Programming primarily is characterized by two features:
  - Inheritance (Chapter 9)
    - New classes created from (extends) existing classes by absorbing (*inheriting*) their attributes and behaviors
  - Polymorphism (Chapter 10)
    - Allows programmers to write applications in a *general* fashion to handle a wide variety of *existing* and *new* problems

3 **Inheritance**

- When a new class is created, it *inherits* any variables and methods of a previously defined superclass
- This subclass gets its *initial* features from the direct superclass
- An indirect superclass is inherited from *two or more* levels above in a class hierarchy

4 **The Subclass**

- The subclass is usually *larger* (logically) than its superclass …
  - *Adds* instance variables and methods of its own to the superclass
  - Also it is possible to define *additions* to, or *replacements* for, inherited superclass features
- It is more *specific* than the superclass …
  - Therefore it has a smaller number of situations in which it can be used

5 **The Superclass**

- Each superclass exists at the top of a *hierarchical* relationship with its subclasses
- A superclass may have *several* direct subclasses which inherit its features
- A subclass *to one* superclass may be a superclass *to other* subclasses

7 **The Keyword super**

- Subclass methods usually reference public and protected members by name
- To reference a member of a superclass that is *overriden* in subclass, use keyword super
- Format:

  super.*member*
- Examples:

```
super(n); // Call superclass constructor
setTuition(super.getCredits());
super.toString();
```

8 **The Keyword extends (Page 1)**

- A new class is *never* created "from scratch"
  - Uses pieces of existing classes

- The keyword extends is used to *inherit* the existing elements from another the class directly above (its direct superclass)

9 **The Keyword extends (Page 2)**

- Format:
  public class *SubClassName* extends *SuperClassName* {
- Examples:
  public class Time extends Object {
  public class SuffolkResident extends Student {

10 **Inheriting from the class Object**

- The class Object has 11 previously defined methods
- In the statement:
  public class Time extends Object {
  – Object is the super-class (base class)
  – Time is the subclass (derived class)
- All classes in Java are subclasses of Object (directly or indirectly) and therefore inherit all of its features

11 **Inheriting from Other Superclasses**

- A subclass inherits the elements of its superclass (methods and instance variables)
- In the statement:
  public class SuffolkResident extends Student {
  – Student is the super-class
  – SuffolkResident is the subclass
- SuffolkResident inherits all of methods *and* instance variables of Student

12 **Inheritance Example (Page 1)**

- Consider the following superclass and subclass declarations:

```
public class Student extends Object {
    private String firstName;
    private String lastName;
    private String grade;
    private int credits;
            ...
public class SuffolkResident extends Student {
    private int tuition;
```

13 **Inheritance Example (Page 2)**

- Effectively instance variables of new objects instantiated from SuffolkResident are:

```
public class SuffolkResident {
    private String firstName;
    private String lastName;
    private String grade;
```

```
private int credits;
private int tuition;
```

- Objects instantiated from the subclass inherit *instance variables* of the superclass Student

14 **Inheritance Example (Page 3)**

- Additionally all *methods* of the superclass Student become part of the subclass SuffolkResident …
  - Only exception are methods of the superclass that are *overridden* by the subclass object
    - The two methods have the same name
  - I.e. the toString() method of the subclass will be called if it *exists in both* the superclass and any of its subclasses

17 **Standard Features of Subclasses**

- The subclass constructor calls its direct superclass constructor
  - This call may be either *explicit* or *implicit*
  - Initializes the instance variables that it inherits
- The subclass toString() method *overrides* toString() method of its direct superclass
  - Or *adds* to superclass toString() method by *extending* (concatenating) its string output onto the superclass' toString() output

19 **Downcasting and Polymorphic Behavior (Page 1)**

- Casting a superclass reference to a subclass reference
- Technique makes it possible to reference a subclass method from an object instantiated from its superclass
- Accomplished by casting the superclass object to the subclass type

20 **Downcasting and Polymorphic Behavior (Page 2)**

- Format:

  SuperClassName object = new *SubClassConstructor*([*args*]);
  - Possible only because the subclass is *derived* (extends from) from the superclass

21 **Downcasting and Polymorphic Behavior (Page 3)**

- Examples:

```
Student s1 = new SuffolkResident("Sally", "Walters", "Z", 7);
Student s2 = new NassauResident("Charles", "Jones", "B+", 13);
…
JOptionPane.showMessageDialog(null, s1.getTuition() )
```

  - Calls getTuition() method of SuffolkResident class (not of subclass NassauResident or superclass Student)

23 **No-Argument Constructors and Inheritance**

- Superclasses and subclasses may have overloaded constructors
  - It usually is up to the programmer to ensure matching of parameters between and

among the constructor for these classes

- These overloaded constructors may include no-argument constructors
- When a subclasses constructor is called, it will make an implicit call to the default no-argument constructor if there is no explicit constructor call