

Inheritance

Inheritance

- Many objects have a **hierarchical** relationship
 - Examples: zoo, car/vehicle, card game, airline reservation system
- **Inheritance** allows software design to take advantage of relationships, supporting reuse
- Supports the IS-A relationship
 - what's the HAS-A relationship?

Terminology

- Base class/Parent class/Superclass
 - defines generic functionality
- Derived class/Child class/Subclass
 - **extends** or **specializes** base class

Syntax

```
public class Student extends Person {...}  
public class Derived extends Base{...}
```

```
public class Person {  
    ...  
    void print();  
}  
  
public class Student extends Person {  
    ...  
    void print();  
}
```

Subclasses

- **Inherit** members of parent
 - May implement new members
 - May **override** members of parent
- | | |
|------------------|---------------------------|
| • Person | • Student |
| – name | – major |
| – Person(String) | – Student(String, String) |
| – print() | – print() |
| – getName() | – changeMajor(String) |

Method Invocation

- Person – print, getName
- Student – print, changeMajor

```
Person p = new Person("Bob");  
Student s = new Student("Alice", "CS");  
s.getName();  
p.print();  
s.print();  
p.changeMajor("Econ");  
s.changeMajor("Econ");
```

Method Invocation

- Person – print, getName
- Student – print, changeMajor

```
Person p = new Person("Bob");
Student s = new Student("Alice", "CS");
s.getName(); //Person getName
p.print(); //Person print
s.print(); //Student print
p.changeMajor("Econ"); //ERROR
s.changeMajor("Econ"); //Student changeMajor
```

Subclasses

- | | |
|------------------|---------------------------|
| • Person | • Student |
| – name | – major |
| – Person(String) | – Student(String, String) |
| – print() | – print() |
| – getName() | – changeMajor(String) |

Protected

- private members of parent not accessible to child class
- **protected** members accessible only to derived classes
- examples

```
public class Person {
    //will be accessible in Student
    protected String name;
}
```

Partial Overriding

```
//in Person class
void print() {...}

//in Student class
void print() {
    super.print();
    //super.methodName();
    ...
}
```

More on *super*

```
public Person(String name) {
    this.name = name;
}

public Student(String name, String major) {
    super(name);
    //super(params);
    this.major = major;
}
```

Shadowing Variables

- Variable (same name) declared in base class and derived class
- Generally, a bad idea
- Example: Student declares a String variable *name*

final

- Classes and methods can be defined with final modifier
 - final public class Student ...
 - final public void print()...
- final classes cannot be extended
- final methods cannot be overridden

abstract

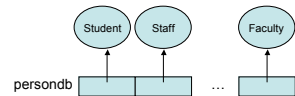
- abstract classes cannot be instantiated
- Declare abstract class using abstract keyword
 - public abstract class Person ...
- Method can also be abstract
 - public abstract void print()
- A class with an abstract method must be declared abstract

Polymorphism

- *Many forms*
- A variable is **polymorphic** if it can refer to different types of objects at different times

```
Person p = new Person("Bob");  
p = new Student("Sally", "History");
```

Example



```
Person[] persondb = new Person[10];  
persondb[0] = new Student("Sally", "History");  
persondb[1] = new Staff(...);  
...  
persondb[9] = new Faculty(...);
```

Dynamic Binding

- Determine which method is called based on object contents

```
class Person {  
    void print();  
}  
class Student extends Person {  
    void print();  
}
```

Dynamic Binding

```
Person p = new Person(...);  
Student s = new Student(...);  
p.print(); //calls Person print()  
p = s; //OKAY  
p.print(); //calls Student print()  
p.changeMajor(); //ERROR
```

Casting

```
Person p;  
p = new Student(...);  
Student s = (Student)p;  
  
if(p instanceof Student)  
    s = (Student)p;
```

- If cast is not successful, runtime error `ClassCastException`
- `instanceof` operator used to determine type

Example

Class1	Class2 extends Class1
1. f1	3. f2
2. f2	4. f3

```
Class1 c1 = new Class2();  
Class2 c2 = new Class2();  
c1.f3();  
c1.f2();  
c2 = (Class2)c1;  
c2.f3();
```

Object base class

- All classes derive from `Object`
- Defines several methods that can be overridden
 - `String toString()`
 - `boolean equals(Object)`
 - `Object clone()`