Administrivia

- Student hours (or by appointment):
  - Listed on Canvas in Syllabus
  - MTWTh 10:30 – 11:30 a.m. via Zoom
- Assignment five posted
  - Due Friday 14 July by **10pm**

CptS 355- Programming Language Design

Object Oriented Programming and Object Oriented Languages

Instructor: Jeremy E. Thompson

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

5 Insights About OO Programming

1. Data and operations belong *together*
2. Abstraction vs. interface: implementation *details* are hidden
3. Subtyping
4. Dynamic Dispatch
5. Inheritance

5 insights about OO Programming

1. Data and operations belong together
   - By packaging data and operations/methods
     - clearer and easier to write and understand programs
     - *legal* operations for data are more easily enforced
   - This is closely related to "*data abstraction*" or "*abstract data types*"

5 insights about OO Programming

2. Abstraction and interface: implementation *details are hidden*
   - access to the data should be only via the defined (*abstract*) operations

- **Interface**: a set of operations expressed in *application-level* terms rather than *implementation-level* terms
  - Includes: operations, their arguments, their results, and their *meaning*
- In Java, the use of the term "*interface*" is little bit different
  - a reference type in Java
  - collection of *abstract* methods—a class *implements* an *interface*
  - along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types
  - unless the class that implements the interface is *abstract*, all methods of the interface must be defined in the class

5 insights about OO Programming

3. Subtyping
   - A is a subtype of type B when:
     - methods (A) $\supseteq$ methods (B), and
     - fields(A) $\supseteq$ fields (B)
   - Notation: **A<:B**
   - If A<:B then:
     - A can be used whenever B can

- If A<:B, then "A *provides all operations* B provides, in terms of ***signatures***."
  - However, the operations *may not have the same meaning*
  - Consequently, languages require programmer to *explicitly declare* subtype relationships

5 insights about OO Programming

4. *Dynamic* and *Static* Dispatch (polymorphism)
   - Dynamic Dispatch:
     - When we have *subtyping*, the method to call depends on the *actual value* contained in variable, *not* its type
     - Binding of the method is determined at run time depending on the type of the object pointed to

Dynamic vs Static Dispatch

- **C++ Example**

```
class Person
{
  public:
    void setSSN(std::string myssn){
        ssn=myssn;
    }
    void print() {
       std::cout << ssn << std::endl;
    }
  private:
    std::string ssn;
};
```

```
class Student : public Person
{
  public:
    void setGPA(float mygpa){
        gpa=mygpa;
    }
    void print() {
       std::cout << gpa << std::endl;
    }
  private:
    double gpa;
};
```

```
1  int main(){
2      Student s ;
3      s.setSSN("999-99-9999");
4      s.setGPA(3.41);
5      Person *p = &s;
6      s.print();
       (*p).print();
   }
```

The *compile-time* (static) type of (*p) is Person
The *run-time* (dynamic) type of (*p) is Student

Dynamic Dispatch

```
int main(){
    Student s ;
    s.setSSN("999-99-9999");
    s.setGPA(3.41);
    Person *p = &s;
    s.print();
    (*p).print();
}
```

- In C++, by *default*, the decision on which member function to invoke (base or overridden) is made on the basis of the *compile-time* type
  - This is called *static* dispatch
- In C++, the decision is made based on the run-time type when the member function is defined as a *virtual* function
  - This is called *dynamic* dispatch

Dynamic Dispatch

- **C++ Example**

```
class Person
{
  public:
    string& setSSN(std::string myssn){
        ssn=myssn;
    }
    virtual void print() {
        std::cout << ssn << std::endl;
    }
  private:
    std::string ssn;
};
```

```
class Student : public Person
{
  public:
    string& setGPA(float mygpa){
        gpa=mygpa;
    }
    virtual void print() {
        std::cout << gpa << std::endl;
    }
  private:
    double gpa;
};
```

```
1  int main(){
2      Student s ;
3      s.setSSN("999-99-9999");
4      s.setGPA(3.41);
5      Person *p = &s;
6      s.print();
       (*p).print();
   }
```

- A *non-virtual* function uses static dispatch
- A *virtual* function uses dynamic dispatch
- here, `(*p).print()` uses *gpa not ssn*

Dynamic Dispatch

```
int main(){
    Student s ;
    s.setSSN("999-99-9999");
    s.setGPA(3.41);
    Person *p = &s;
    s.print();
    (*p).print();
}
```

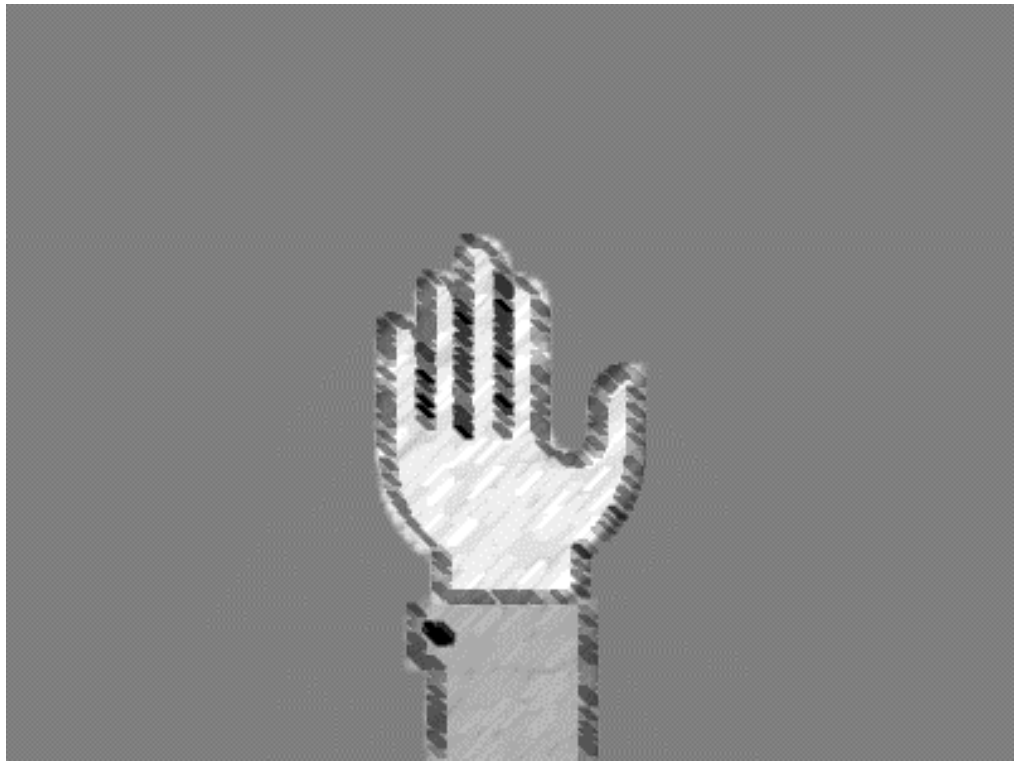
- In C++, by *default*, the decision on which member function to invoke (base or overridden) is made on the basis of the *compile-time* type
  - This is called *static* dispatch
- The decision is made based on the run-time type, when the member function is defined as a *virtual* function
  - This is called *dynamic* dispatch
- How about Java?
- Why is static dispatch the default in C++?

5 insights about OO Programming

5. Inheritance
   - The *implementations* of methods of a supertype are available (and used) in a subtype *unless* they are overridden

- Note: Subtyping vs inheritance
  - subtyping is about *compatibility* of *interfaces* (in the general sense)
  - inheritance is about *re-use* of *implementations*

Questions?



RECALL: Dynamic Dispatch

```
int main(){
    Student s ;
    s.setSSN("999-99-9999");
    s.setGPA(3.41);
    Person *p = &s;
    s.print();
    (*p).print();
}
```

- In C++, by *default*, the decision on which member function to invoke (base or overridden) is made on the basis of the *compile-time* type
  - This is called *static* dispatch
- The decision is made based on the run-time type, when the member function is defined as a *virtual* function
  - This is called *dynamic* dispatch
- How about Java?
- Why is static dispatch the default in C++?

How are *virtual* methods implemented in C++?

- On a *per-class* basis (not *per-instance*)
  - run-time data structure called a *v-table* that contains pointers to the code for *virtual* methods

How are objects and virtual methods implemented in C++?

- Example:

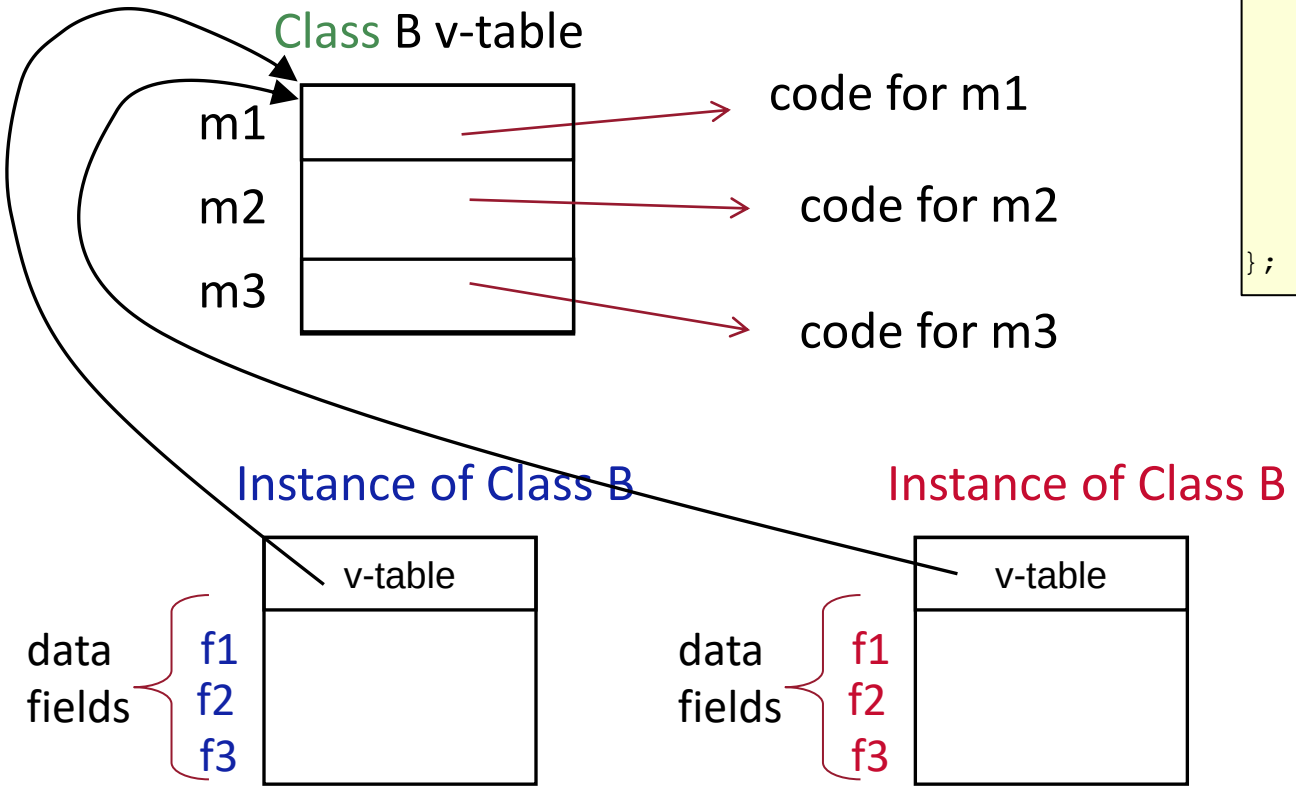
```
class B
{
  public:
    int f1;
    int f2;
    int f3;
    virtual void m1{ some code  }
    virtual void m2{ some code  }
    virtual void m3{ some code  }
    void m4{ some code }
};
```

```
class A : public B
{
  public:
    int f4;
    virtual void m2{ some code  }
    virtual void m3{ some code  }
    virtual void m5{ some code  }
    void m4{ some code }
};
```

```
int main(){
1       B p ;
2       p.m1();
3       B *q = new A();
4       q->m1();
5       q->m2();
6       q->m4();
}
```

How are objects and virtual methods implemented in C++?

- *v-table:* run-time data structure which contains pointers to the code for the virtual methods




```
class B
{
  public:
    int f1;
    int f2;
    int f3;
    virtual void m1{ some code  }
    virtual void m2{ some code  }
    virtual void m3{ some code  }
    void m4{ some code }
};
```

```
int main(){
    B x ;
    x.m1();
    B *y = new B();
    y->m2();
}
```

Since m4 is a non-virtual method, what happens?

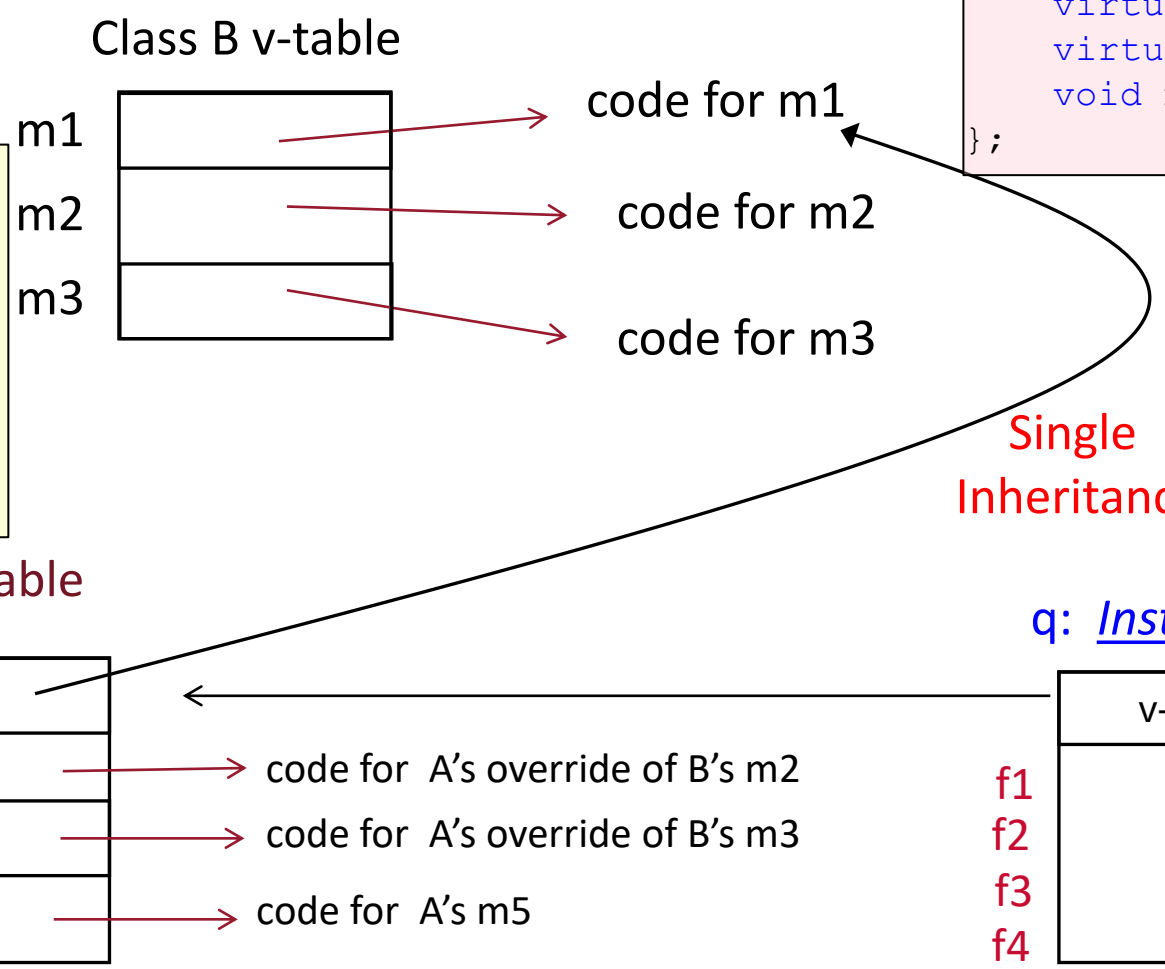
How are objects and virtual methods implemented in C++? (cont)

- class A is *subclass* of class B

```
class A : public B
{
  public:
    int f4;
    virtual void m2{ some code  }
    virtual void m3{ some code  }
    virtual void m5{ some code  }
    void m4{ some code }
};
```

```
class B
{
  public:
    int f1;
    int f2;
    int f3;
    virtual void m1{ some code  }
    virtual void m2{ some code  }
    virtual void m3{ some code  }
    void m4{ some code }
};
```

```
int main(){
    B p ;
    p.m1();
    B *q =  new A();
    q->m1();
    q->m2();
    q->m4();
}
```



How are objects and methods implemented in Java?

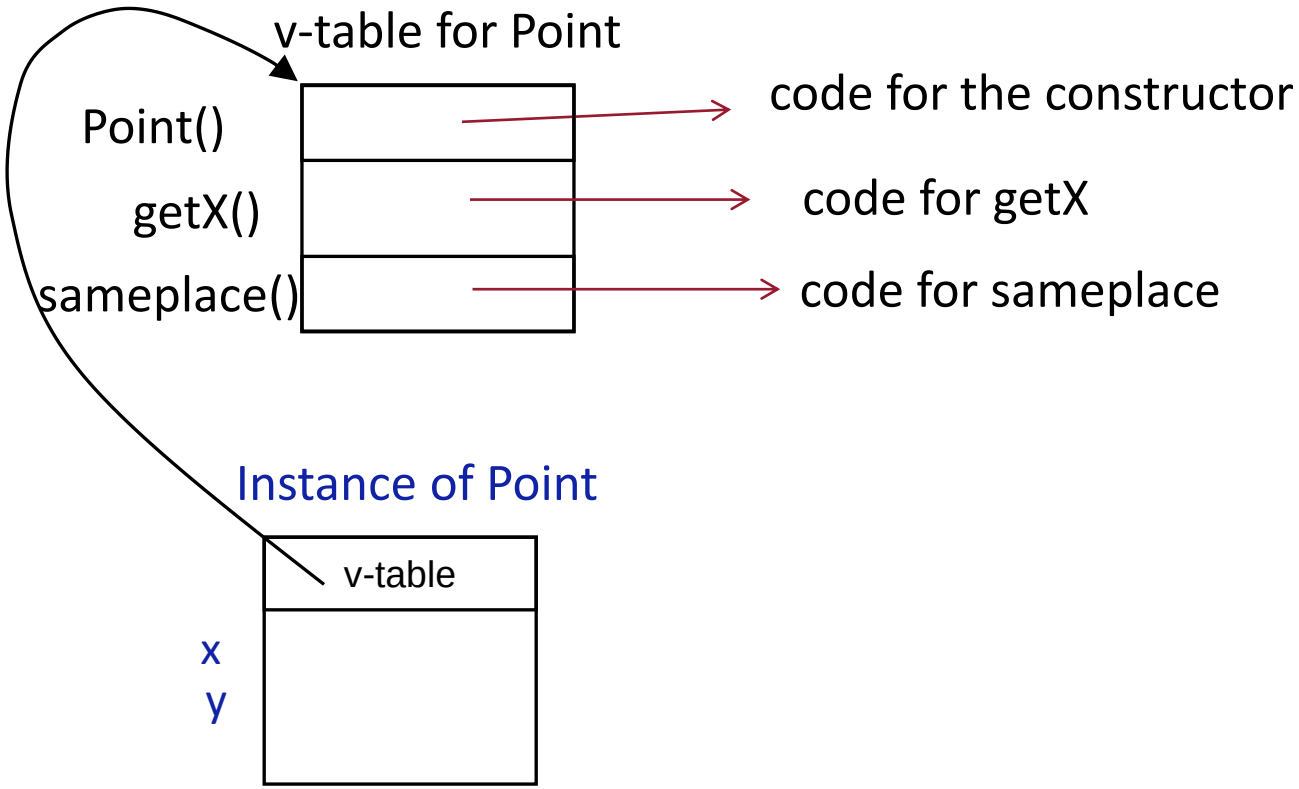
- Example:

```
class Point
{
  public:
    double x;
    double y;
    Point (double x,double y){
       this.x = ; this.y=y;
    }
    double getX(){
       return x;
    }
    boolean sameplace (Point p){
       return (x==p.x) && (y==p.y)
    }
}
```

```
class PtSubClass extends Point
{
 public:
    int aNewField;
    PtSubClass(double x,double y){
       super(x,y);
    }
    boolean sameplace (Point p){
       return false;
    }
    void sayHi () {
       System.out.println("hello!");
    }
}
```

```
int main(){
     Point p = new Point();
     Point q = new PtSubClass ();
     ...
}
```

How are objects and methods implemented in Java?




```
class Point
{
  public:
    double x;
    double y;
    Point (double x,double y){
       this.x = ; this.y=y;
    }
    double getX(){
       return x;
    }
    boolean sameplace (Point p){
       return (x==p.x) && (y==p.y)
    }
}
```

- V-table is shared across *all objects* in class!
- If the object instance of Point is no longer needed, what will happen to its v-table?

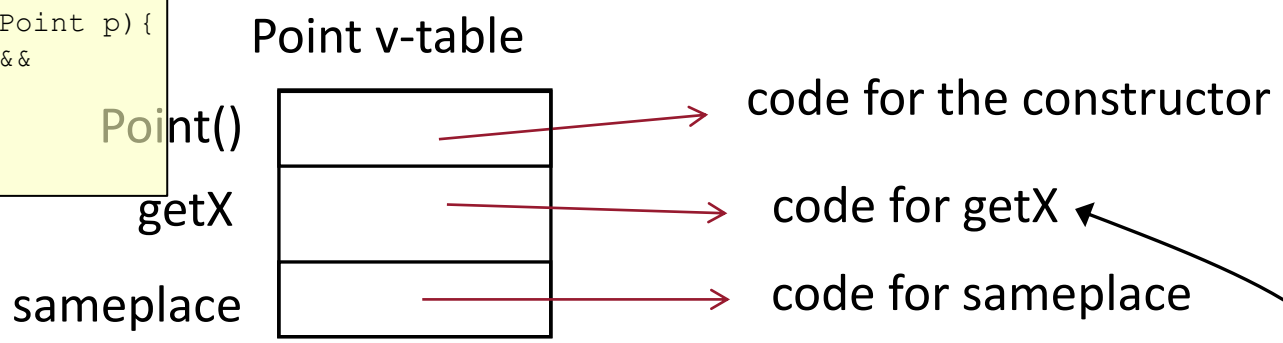
How are objects and methods implemented in Java? (cont)

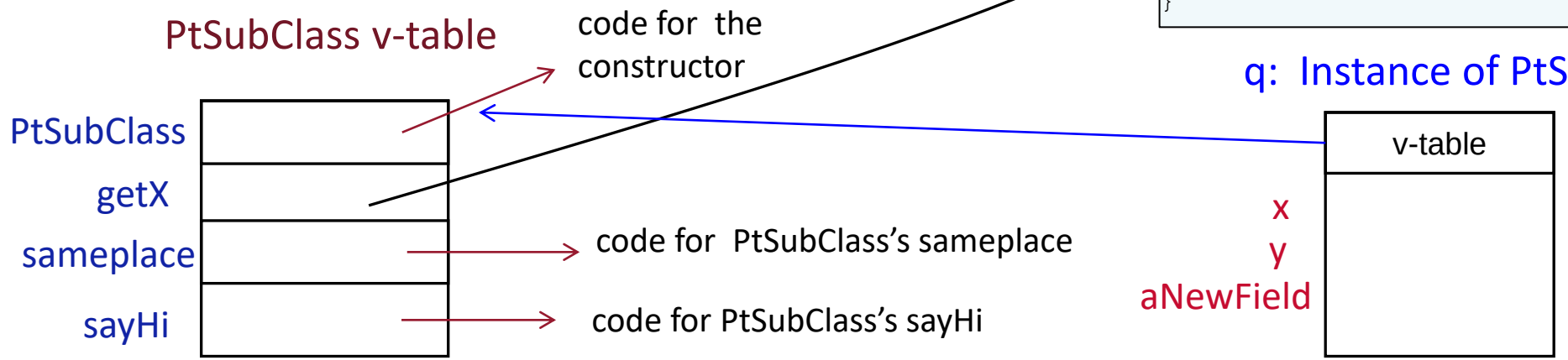
```
class Point
{
  public:
    double x;
    double y;
    Point (double x,double y){
       this.x = ; this.y=y;
    }
    double getX(){
       return x;
    }
    boolean sameplace (Point p){
       return (x==p.x) &&
(y==p.y)
    }
}
```

- PtSubClass is subclass of Point

```
class PtSubClass extends Point
{
 public:
   int aNewField;
   PtSubClass(double x,double y){
      super(x,y)
   }
   boolean sameplace (Point p){
      return false;
   }
   void sayHi () {
    System.out.println("hello!");
   }
}
```

```
int main(){
    Point p = new Point();
    Point q = new PtSubClass ();
    ...
}
```





Questions?