

CS 213 : Software Methodology

Fall 2020

Sesh Venugopal

Inheritance: Private Fields/Static Members

Inheritance - Private Fields

```
public class Point {  
    private int x,y;  
    ...  
}
```

```
public class ColoredPoint extends Point {  
    // x and y inherited but HIDDEN  
    ...  
    public int getX() { // override inherited getX()  
        return x;  
    }  
}
```

COMPILE?

WILL NOT COMPILE
because `x` is hidden



Inheritance - Private Fields

```
public class Point {  
    private int x,y;  
    ...  
}  
  
public class ColoredPoint extends Point {  
    // x and y inherited but HIDDEN  
    ... // getX() is NOT overridden  
}
```

```
public class PointApp {  
    public static void  
    main(String[] args) {  
        ColoredPoint cp = new ColoredPoint(4,5,"blue");  
  
        System.out.println(cp.x); // ? WILL NOT COMPILE, x is hidden  
        System.out.println(cp.getX()); // ? 4  
  
        Inherited getX() method is  
        able to access the x field  
  
    }  
}
```

Inheritance - Static Members

```
public class Supercl {  
    static int x=2;  
    public static void m() {  
        System.out.println(  
            "in class Supercl");  
    }  
}
```

```
public class Subcl  
    extends Supercl { }
```

```
public class StaticTest {  
    public static void main(String[] args) {  
        System.out.println(Supercl.x); // ? 2  
        Supercl.m(); // ? "in class Supercl"  
        System.out.println(Subcl.x); // ? 2 – inherited from Supercl  
        Subcl.m(); // ? "in class Supercl" – inherited from Supercl  
    }  
}
```

Inheritance - Static Fields

```
public class SuperCl {  
    static int x=2;  
    public static void m() {  
        System.out.println("in class SuperCl");  
    }  
}
```

```
public class SubCl  
extends SuperCl {  
    int x=3;  
}
```

↑
Instance field with
same name as
inherited static field x

```
public class StaticTest {  
    public static void main(String[] args) {  
        System.out.println(SubCl.x); // ? DOES NOT COMPILE  
    }  
}
```

“cannot make static reference to non-static field x”

Instance field of same name will HIDE inherited static field

Inheritance - Static Fields

```
public class Supercl {  
    static int x=2;  
    public static void m() {  
        System.out.println("in class Supercl");  
    }  
}  
  
public class Subcl  
    extends Supercl {  
    int x=3;  
}  
  
public class StaticTest {  
    public static void main(String[] args) {  
        Subcl subclref = new Subcl();  
        System.out.println(subclref.x); // ? 3 – instance field x  
    }  
}
```

Dynamic Binding

```
public class PointApp {  
    public static void  
    main(String[] args) {  
  
        Point p3 = new ColoredPoint(2,3,"red");  
  
        System.out.println("p3 = " + p3); // ? "p3 = 2,3,red"  
  
    }  
}
```

↑ static type ↑ dynamic type

Dynamic Binding

Static type of p3 is `Point`,
but dynamic type (type of
instance it points to) is
`ColoredPoint`.



So, the `p3.toString()`
static call is bound to the
dynamic type,
`ColoredPoint`.



This results in the
overriding version
of `toString()` in
`ColoredPoint` being
executed.

Inherited Static Field Binding

```
public class Supercl {  
    static int x=2;  
    public static void m() {  
        System.out.println("in class Supercl");  
    }  
}
```

```
public class Subcl  
extends Supercl {  
    int x=3;  
}
```

```
public class StaticTest {  
    public static void main(String[] args) {
```

```
        Supercl superclref = new Subcl();
```


static type


dynamic type

```
        System.out.println(superclref.x); // ? 2 – inherited static field x !!!
```

```
    }  
}
```

INHERITED STATIC FIELDS ARE STATICALLY BOUND (TO REFERENCE/STATIC TYPE), NOT DYNAMICALLY BOUND (TO INSTANCE/DYNAMIC TYPE) –

Inherited Static Method Binding

```
public class Sorter {  
    public static void  
    sort(String[] names) {  
        System.out.println(  
            "simple sort";  
        }  
    }  
}
```

```
public class IllustratedSorter  
    extends Sorter {  
    // override  
    public static void  
    sort(String[] names)  
        System.out.println(  
            "illustrated sort";  
        }  
    }  
}
```

```
Sorter p = new IllustratedSorter();
```

↑
static type

↑
dynamic type

```
p.sort(); // ? "simple sort"
```

`sort()` is statically bound to `p`, meaning
since `Sorter` is the reference/static type of `p`,
the `sort()` method in `Sorter` is called

Inherited Static Method Binding

```
public class Sorter {  
    public static void  
    sort(String[] names) {  
        System.out.println(  
            "simple sort";  
        }  
    }  
}
```

```
public class IllustratedSorter  
    extends Sorter {  
    // override  
    public static void  
    sort(String[] names)  
        System.out.println(  
            "illustrated sort";  
        }  
    }  
}
```

```
Sorter p = new IllustratedSorter();
```

↑
static type

↑
dynamic type

```
p.sort(); // ? "simple sort"
```

`sort()` is statically bound to `p`, meaning
since `Sorter` is the reference/static type of `p`,
the `sort()` method in `Sorter` is called