

Lecture 2

Java OOP Revisited

Package in Java

- A package is mapped to a directory

`utilities/num` → `utilities.num`

`java/util` → `java.util`

- Fully qualified class name

- FQN = *package_name.class_name*

- Uniquely identifies a class

`vn.hanu.fit.jsd.HelloWorldApp`

Java CLASSPATH

- Specifies directories for JVM to find classes
- Default value is “.” which is the current directory
- Specified using the JVM option -cp or -classpath

```
java -cp "D:\area.jar;."
```

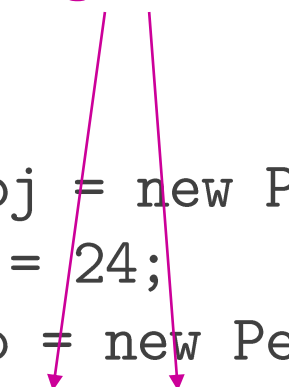
- Can be set in the CLASSPATH environment variable
(not recommended)

Java Method

- When a method is called:
 - argument values (not arguments) are passed
 - argument and corresponding parameter are two separate variables
- Affects primitive and reference-type arguments differently (how?)

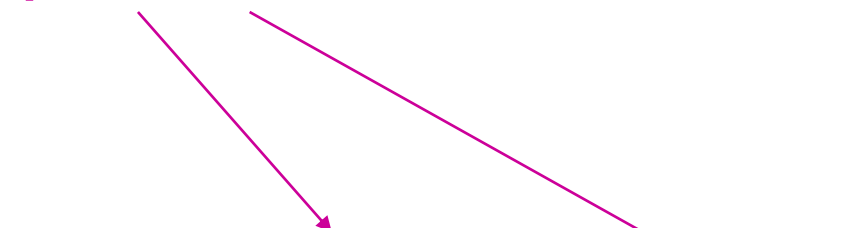
arguments

```
PList obj = new PList();  
int age = 24;  
Person p = new Person();  
obj.add(age, p);
```



parameters

```
class PList {  
    public void add(int age, Person p) {  
    }  
}
```



this parameter

- An implicit (hidden) parameter available to all methods of a class
- Used to explicitly refer to other instance variables and methods
- The current object is plugged in for this

```
public void setDate(int month, int day, int year) {  
    this.month = monthString(month);  
    this.day = day;  
    this.year = year;  
}
```

Information hiding and encapsulation

- **Information hiding:** hide internal design details of a class from others
 - a.k.a. separate implementation details from specification
 - modification of implementation does not affect client code
- **Encapsulation:** put data and methods that operate on that data in a single unit
 - Task: to design suitable interface methods of a class
 - observers, mutators, constructors...

Access modifiers

- Access modifiers: `public`, `private`, `protected`
- `public`: no restrictions on where a member (variable or method) may be used
 - used for interface methods
- `private`: cannot be accessed outside of its class
 - used for instance variables
- `protected`: accessible from inside and from sub-classes
- Omitted: default (package) access, accessible from inside class, and other classes in the same package

Copy constructor

- A constructor that creates an object using another object of the same class

```
public class Employee {  
    private int id;  
    private String name;  
    public Employee(Employee employee) {  
        this.id = employee.id;  
        this.name = employee.name;  
    }  
}
```


final class & methods

- A `final` class is a class that can't be extended
- A `final` method is a method that can't be overridden by subclasses

Immutable Class

- Class is declared as `final` so that it can't be extended
- All fields should be `private` to block direct access to them
- No mutators (setters)
- All mutable fields declared as `final` (constant)
- Immutable classes are thread-safe
- Thread safety
 - A class is thread-safe if its instances functions correctly during simultaneous execution by multiple threads

Other concepts

- Class, variables, methods
- Static members
- Comments & javadoc
- Preconditions and postconditions
- Method overloading
- Object cloning
- Mutable and immutable classes

Inner class

- A class defined within another class
 - the enclosing class is the outer class
- Considered a member of the outer class
- Used as a helper class
 - implement some aspects of the outer class
- Has full access to members of the outer class
 - and vice versa

Inner class example

```
public class BankAccount {  
    private Money balance;  
    public BankAccount() {  
        balance = new Money("0.00");  
    }  
    public String getBalance() { return  
        balance.getAmount();  
    }  
    private class Money {  
        private long dollars;  
        private int cents;  
        public Money(String stringAmount) {  
            // ...code omitted...  
        }  
    } // end Money  
}
```

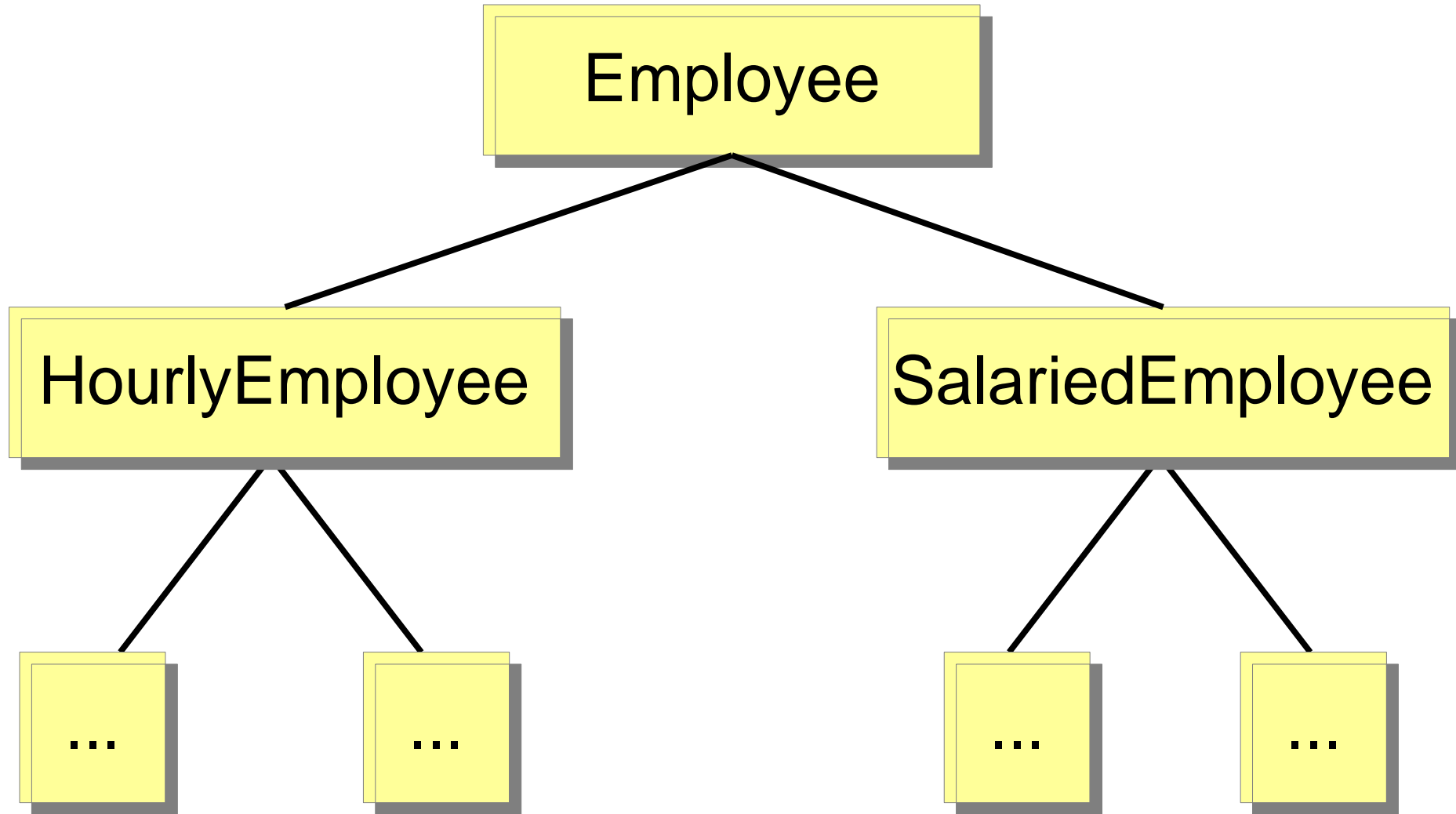
Inner class usages

- Static inner class
- Public inner class
- Nesting inner class

Inheritance

- A new (sub) class is derived from another (super) class
 - superclass is the base class
- Subclass has (or inherits) all members of the superclass
- Subclass access non-static members of superclass using `super`
- A subclass object has multiple types:
 - subclass, superclass, and other ancestors of the superclass

Inheritance hierarchy



Inheritance & encapsulation

- private members are inherited but not directly accessible in the subclass
- Example:

```
joe.setName("Josephine"); // ok
```

```
joe.getName(); // ok
```

```
joe.name = "Josephine"; // error
```

- To allow access in a subclass, use protected
 - for setters/getters (rather than for variables)

Method overriding

- Access modifier can be “wider” but not “narrower”
 - e.g. private → default, protected or public, etc.

```
public class BaseClass {  
    protected Employee getSomeone(int someKey) {  
        // code omitted  
    }  
}
```

```
public class DerivedClass extends BaseClass {  
    private HourlyEmployee getSomeone(int someKey){  
        // code omitted  
    } // error  
}
```

Constructor overloading

- Constructors of the same class can overload
- Overloading constructor invokes another using `this`
 - must be the first line

```
// default constructor  
public HourlEmployee() {  
    this("No name", new Date("January", 1, 1000), 0,0);  
}
```

The super constructor

- May be invoked by a constructor of a subclass
- The default constructor, `super()`, is invoked by default
- Other constructors need to be invoked explicitly if required
- The `super` call must be the first line in the subclass constructor
- Cannot use subclass' instance variables as arguments

Polymorphism (i)

- Applied to classes in an inheritance hierarchy
- Means “multiple forms”:
 - a superclass variable can refer to any subclass object
- Used to define *polymorphic methods*:
 - parameter type is a superclass
 - invoked with a subclass object

Polymorphic method example (1)

```
public class Sale {  
    private String name; // A nonempty string private  
    double price; // nonnegative  
    ...  
    public double bill() {  
        return price;  
    }  
    public boolean lessThan(Sale other) {  
        ...  
        return (bill() < other.bill());  
    }  
    ...  
}
```

Polymorphic method example (2)

```
public class DiscountSale extends Sale {  
    private double discount;  
    ...  
    public double bill() {  
        double fraction = discount / 100;  
        return (1 - fraction) * getPrice();  
    }  
    ...  
}
```

Polymorphic method example (3)

```
public class Sale {  
    private String name; // A nonempty string private  
    double price;        // nonnegative  
    ...  
    public double bill() {  
        return price;  
    }  
    public boolean lessThan(Sale other) {  
        ...  
        return (bill() < other.bill());  
    }  
    ...  
}
```

Polymorphic
method

Polymorphism (ii)

- Polymorphic methods have multiple behaviors
 - depending on the actual type of input object
- *Polymorphic call* is an invocation of a polymorphic method
 - a superclass variable can refer to any subclass object
- Late binding:
 - method definition is determined at run-time based on the actual type of the object invoked
 - Late binding does not apply to `final` methods

Late binding example

```
public class Sale {  
    private String name; // A nonempty string private  
    double price; // nonnegative  
    ...  
    public double bill() {  
        return price;  
    }  
    public boolean lessThan(Sale other) {  
        ...  
        return (bill() < other.bill());  
    }  
    ...  
}
```



late binding

Static binding

- Binding at compile-time (when code is compiled)
- Static binding is applied to these members:
 `private`, `static`, and `final`
- `private` methods are local and thus cannot take different meanings at run-time
- `static` methods are bound to the class and thus cannot take different meanings
- Members or classes marked with `final` cannot be overridden

Other concepts

- Abstract vs. concrete class
- Interface, interface implementation, multiple inheritance
- Anonymous class