

Packages in Java

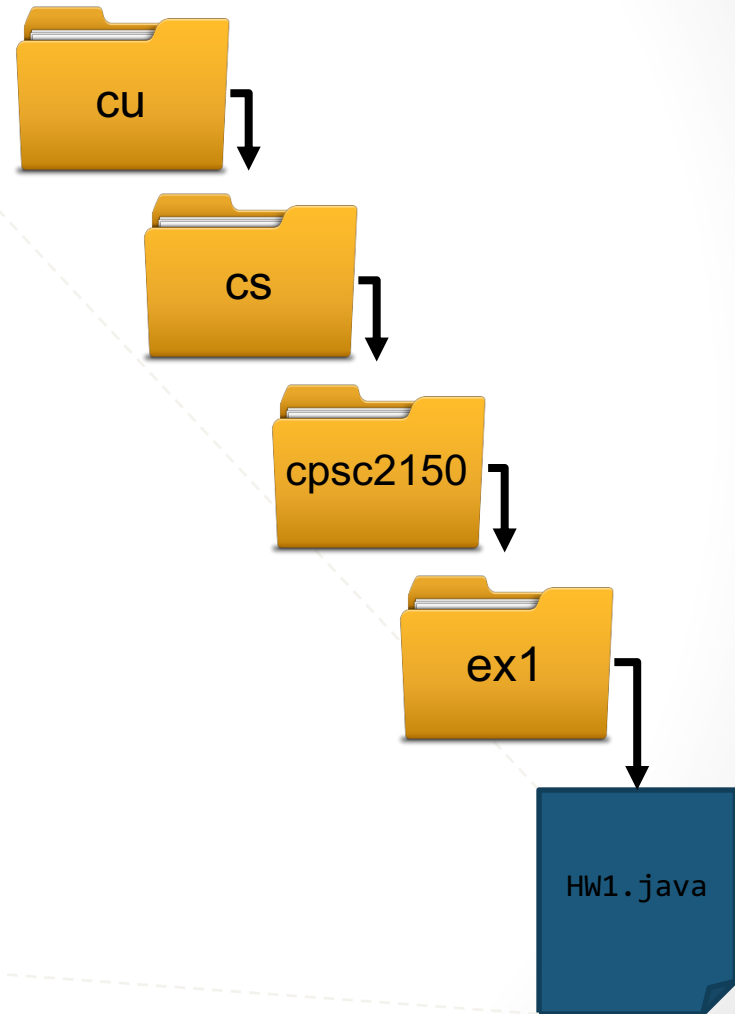
CPSC 2150

Packages: Component Catalogs

- A *package* is a grouping of classes
 - **Hierarchical:** subpackages within packages
 - Java's standard libraries organized in packages
 - `java.lang`, `java.util`, `java.util.logging`
 - <https://docs.oracle.com/en/java/javase/11/docs/api/allpackages-index.html>
- A package provides
 - Logical structuring: related classes are bundled
 - **Encapsulation:** another level of access control
 - **Distinct namespace:** classes in different packages can have the same name without conflict
 - *Convention* to guarantee uniqueness of package name: reverse of company's domain name
 - `org.w3c`, `cu.cs.cpsc2150.ex1`

```
package cu.cs.cpsc2150.ex1;
```

```
class HW1 {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```



Declaration

- Use package statement at top of source file
 - Must appear first, before any class declarations

```
package edu.clemson.cs.cpsc2150;  
public class Pencil { . . . }
```
- This file must be in a directory matching package name
 - Pencil.java in ???/edu/clemson/cs/cpsc2150/
 - IntelliJ handles this correspondence for you
- At most one package declaration in a file
- If there is no package declaration, class is in unnamed default package
 - This is fine only for very small programs

Access Control

- Another level of visibility: package
 - Default for members (`public/private` omitted)
 - Package-visible members are accessible by all classes in the same package

```
package edu.clemson.cs.cpsc2150;  
class Pencil {  
    private String color;  
    int length;  
    . . .  
}
```

- Classes are `public` or package (default)
 - Public classes available outside package
`public class Math { . . . }`
 - Package classes available only within same package
`class Pencil { . . . }`

Type Imports

- Fully-qualified type name is *package.class*
`java.util.Date d = new java.util.Date();`
 - Do not confuse this “.” with member access
- **Shorthand:** `import` statement at top of file
 - To import a single *public* type
`import java.util.Date;`
`Date d = new Date();`
 - To import all *public* types, use wildcard `*`
`import java.util.*;`
`Date d = new Date();`
 - `*` does not import subpackages
- All classes implicitly import `java.lang.*`
- Static members can be explicitly imported
`import static java.lang.Math.exp;`
`exp(x); //instead of Math.exp(x)`
 - Can use wildcard `*` as well

Good Practice: Naming Conventions

- Avoid name conflicts with packages and reserved keywords
- **Package names:** lowercase letters
 - `java.util`, `java.net`, `java.io`, ...
- **Class names:** start with uppercase letter
 - `Math`, `Pencil`, `PriorityQueue`, ...
- **Variable, field and method names:** start with lowercase letters
 - `x`, `out`, `myColor`, `abs()`, `getName()`, `isEven()` ...
- **Constant names:** all uppercase letters
 - `PI`, `DEFAULT_LENGTH`, `DAY_OF_WEEK` ...

Naming Conventions

- `cpssc2150.extendedTicTacToe.GameBoard.MAX_ROWS`
 - From naming conventions we can tell:
 - `cpssc2150` is a package with `extendedTicTacToe` as a subpackage
 - `GameBoard` is a class
 - `MAX_ROWS` is a static constant
- `antonsPizza.DBNinja.getToppings()`
 - From naming conventions we can tell:
 - `antonsPizza` is a package
 - `DBNinja` is a class
 - `getToppings` is a public static function

The end