Packages

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 7

Packages

- Java has an organizational unit called `package`

Packages

- Java has an organizational unit called `package`
- Can use `import` to use packages directly

```
import java.math.BigDecimal
```

Packages

- Java has an organizational unit called `package`
- Can use `import` to use packages directly

  ```
  import java.math.BigDecimal
  ```

- All classes in `.../java/math`

  ```
  import java.math.*
  ```

Packages

- Java has an organizational unit called `package`
- Can use `import` to use packages directly

  ```
  import java.math.BigDecimal
  ```

- All classes in `.../java/math`

  ```
  import java.math.*
  ```

- Note that `*` is not recursive. Cannot write

  ```
  import java.*
  ```

Creating and naming packages

- Can create our own hierarchy of packages

Creating and naming packages

- Can create our own hierarchy of packages
- Naming convention is similar to Internet domain name, but in reverse
  - Internet domain: `onlinedegree.iitm.ac.in`
  - Package name: `in.ac.iitm.onlinedegree`

Creating and naming packages

- Can create our own hierarchy of packages
- Naming convention is similar to Internet domain name, but in reverse
  - Internet domain: `onlinedegree.iitm.ac.in`
  - Package name: `in.ac.iitm.onlinedegree`
- Add a package header to include a class in a package

```
package in.ac.iitm.onlinedegree;

public class Employee { ... }
```

Creating and naming packages

- Can create our own hierarchy of packages
- Naming convention is similar to Internet domain name, but in reverse
  - Internet domain: `onlinedegree.iitm.ac.in`
  - Package name: `in.ac.iitm.onlinedegree`
- Add a package header to include a class in a package

```
package in.ac.iitm.onlinedegree;

public class Employee { ... }
```

- By default, all classes in a directory belong to same anonymous package

More about visibility

- We have seen modifiers `public` and `private`

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package
  - This applies to both methods and variables

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package
  - This applies to both methods and variables
- Can also restrict visibility with respect to inheritance hierarchy

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package
  - This applies to both methods and variables
- Can also restrict visibility with respect to inheritance hierarchy
  - `protected` means visible within subtree, so all subclasses

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package
  - This applies to both methods and variables
- Can also restrict visibility with respect to inheritance hierarchy
  - `protected` means visible within subtree, so all subclasses
  - Normally, a subclass cannot expand visibility of a function

More about visibility

- We have seen modifiers `public` and `private`
- If we omit these, the default visibility is public within the package
  - This applies to both methods and variables
- Can also restrict visibility with respect to inheritance hierarchy
  - `protected` means visible within subtree, so all subclasses
  - Normally, a subclass cannot expand visibility of a function
  - However, `protected` can be made `public`