ITP20003 Java Programming

More About Objects and Methods

(Chapter 6)

This slide is primary taken from the instructor's resource of Java: Introduction to Problem Solving and Programming, 7th ed. by Savitch and then edited partly by Shin Hong

Constructors

- A special method called when instance of an object created with new
  - Create objects
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions
  - Each with different numbers or types of parameters
- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define any constructors
  - If you <u>do</u> define a constructor, Java will <u>not</u> automatically define a default constructor

Static Variables

- Static variables are shared by all objects of a class
  - Variables declared `static final` are considered constants – value cannot be changed
- Variables declared `static` (without `final`) can be changed
  - Only one instance of the variable exists
  - It can be accessed by all instances of the class
- Static variables also called *class variables*
  - Contrast with *instance variables*
- Do not confuse class variables with variables of a class type
- Both static variables and instance variables are sometimes called *fields* or *data members*

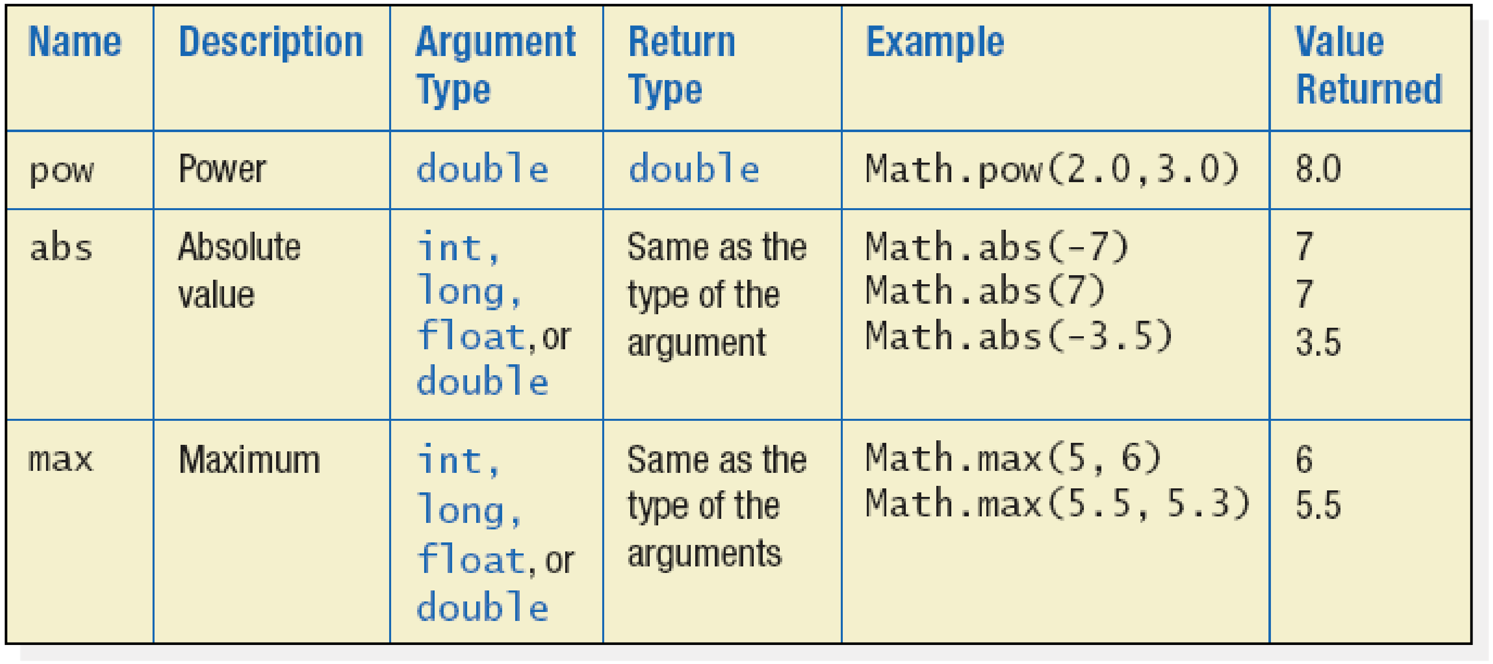
Static Methods

- Static methods are used for constructing functions which are merely working on arguments, rather than on any object instances
  - e.g., compute max of two integers,
    convert character from upper- to lower case

- Static method declared <u>in</u> a class
  - Can be invoked <u>without</u> using an object
  - Instead use the class name

The `Math` Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed

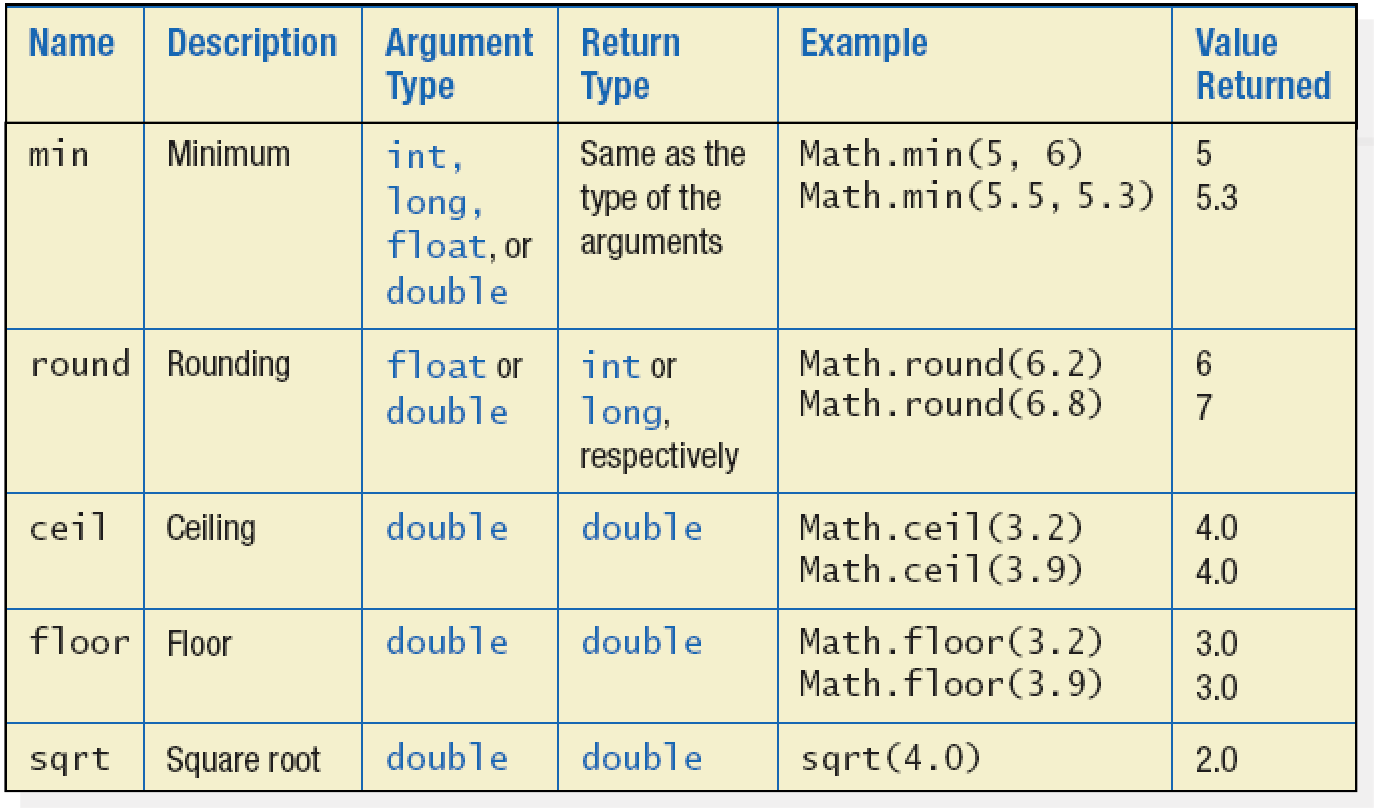


Name	Description	Argument Type	Return Type	Example	Value Returned
`pow`	Power	`double`	`double`	`Math.pow(2.0,3.0)`	8.0

`abs`	Absolute value	`int`, `long`, `float`, or `double`	Same as the type of the argument	`Math.abs(-7)`
				`Math.abs(7)`
				`Math.abs(-3.5)`
					7
					7
					3.5

`max`	Maximum	`int`, `long`, `float`, or `double`	Same as the type of the arguments	`Math.max(5, 6)`
				`Math.max(5.5, 5.3)`
					6
					5.5

The `Math` Class



Name	Description	Argument Type	Return Type	Example	Value Returned

`min`	Minimum	`int`, `long`, `float`, or `double`	Same as the type of the arguments	`Math.min(5, 6)`
				`Math.min(5.5, 5.3)`
					5
					5.3

`round`	Rounding	`float` or `double`	`int` or `long`, respectively	`Math.round(6.2)`
				`Math.round(6.8)`
					6
					7

`ceil`	Ceiling	`double`	`double`	`Math.ceil(3.2)`
				`Math.ceil(3.9)`
					4.0
					4.0

`floor`	Floor	`double`	`double`	`Math.floor(3.2)`
				`Math.floor(3.9)`
					3.0
					3.0

`sqrt`	Square root	`double`	`double`	`sqrt(4.0)`	2.0

Random Numbers

- **`Math.random()`** returns a random double that is greater than or equal to zero and less than 1
  - Java also has a **`Random`** class to generate random numbers
  - Can scale using addition and multiplication; the following simulates rolling a six sided die

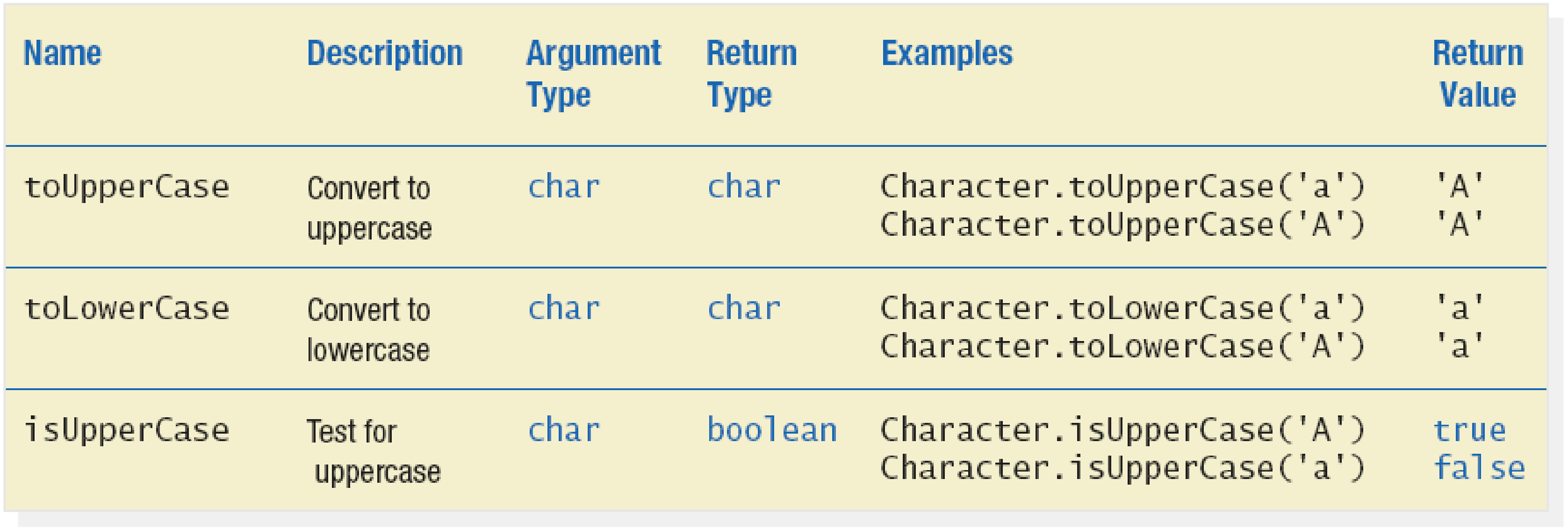
```
int die = (int) (6.0 * Math.random()) + 1;
```

Wrapper Classes

- Java provides *wrapper classes* for each primitive type
- Wrapper classes allow programmer to have an object that corresponds to value of primitive type
  - Methods provided to act on values
  - Contain useful predefined constants and methods
- Wrapper classes have no default constructor
  - Programmer must specify an initializing value when creating new object
- Wrapper classes have no **`set`** methods

Wrapper Classes

- Ex. **`Character`**



Name	Description	Argument Type	Return Type	Examples	Return Value

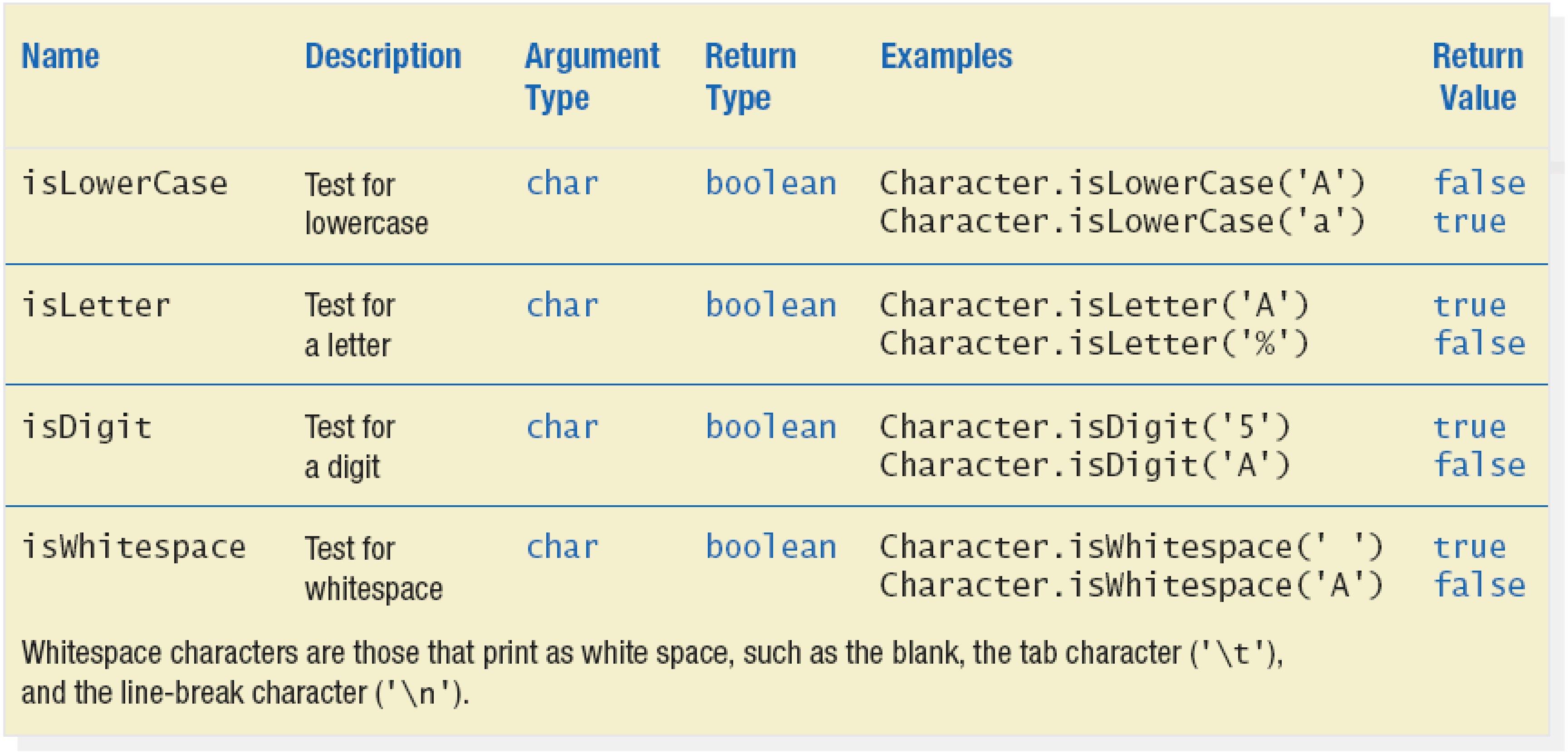
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')`
				`Character.toUpperCase('A')`
					`'A'`
					`'A'`

`toLowerCase`	Convert to lowercase	`char`	`char`	`Character.toLowerCase('a')`
				`Character.toLowerCase('A')`
					`'a'`
					`'a'`

`isUpperCase`	Test for uppercase	`char`	`boolean`	`Character.isUpperCase('A')`
				`Character.isUpperCase('a')`
					`true`
					`false`

Wrapper Classes

- Ex. **`Character`**



Name	Description	Argument Type	Return Type	Examples	Return Value

`isLowerCase`	Test for lowercase	`char`	`boolean`	`Character.isLowerCase('A')`
				`Character.isLowerCase('a')`
					`false`
					`true`

`isLetter`	Test for a letter	`char`	`boolean`	`Character.isLetter('A')`
				`Character.isLetter('%')`
					`true`
					`false`

`isDigit`	Test for a digit	`char`	`boolean`	`Character.isDigit('5')`
				`Character.isDigit('A')`
					`true`
					`false`

`isWhitespace`	Test for whitespace	`char`	`boolean`	`Character.isWhitespace(' ')`
				`Character.isWhitespace('A')`
					`true`
					`false`

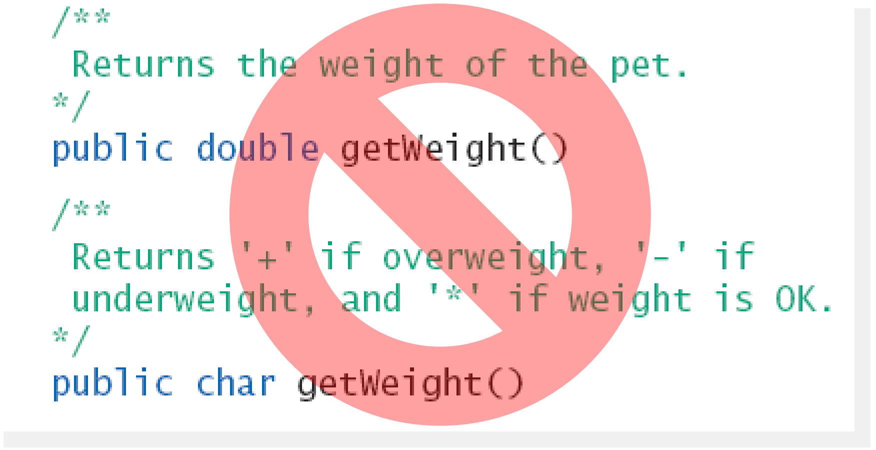
Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line-break character (`'\n'`).

Overloading

- A class can have two or more methods having the same name
- Java distinguishes the methods by number and types of parameters
  - If it cannot match a call with a definition, it attempts to do type conversions
- A method's name and number and type of parameters is called the *signature*

Overloading and Return Type

- You must not overload a method where the only difference is the type of value returned



```
/**
 Returns the weight of the pet.
*/
public double getWeight()

/**
 Returns '+' if overweight, '-' if
 underweight, and '*' if weight is OK.
*/
public char getWeight()
```

Enumeration as a Class

- Consider defining an enumeration for suits of cards
  `enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}`
- Compiler creates a class with methods
  - `equals`
  - `compareTo`
  - `ordinal`
  - `toString`
  - `valueOf`

Enumeration as a Class

- View enhanced enumeration, listing 6.20
  `enum Suit`
- Note
  - Instance variables
  - Additional methods
  - Constructor

Packages: Outline

- Packages and Importing
- Package Names and Directories
- Name Clashes

Packages and Importing

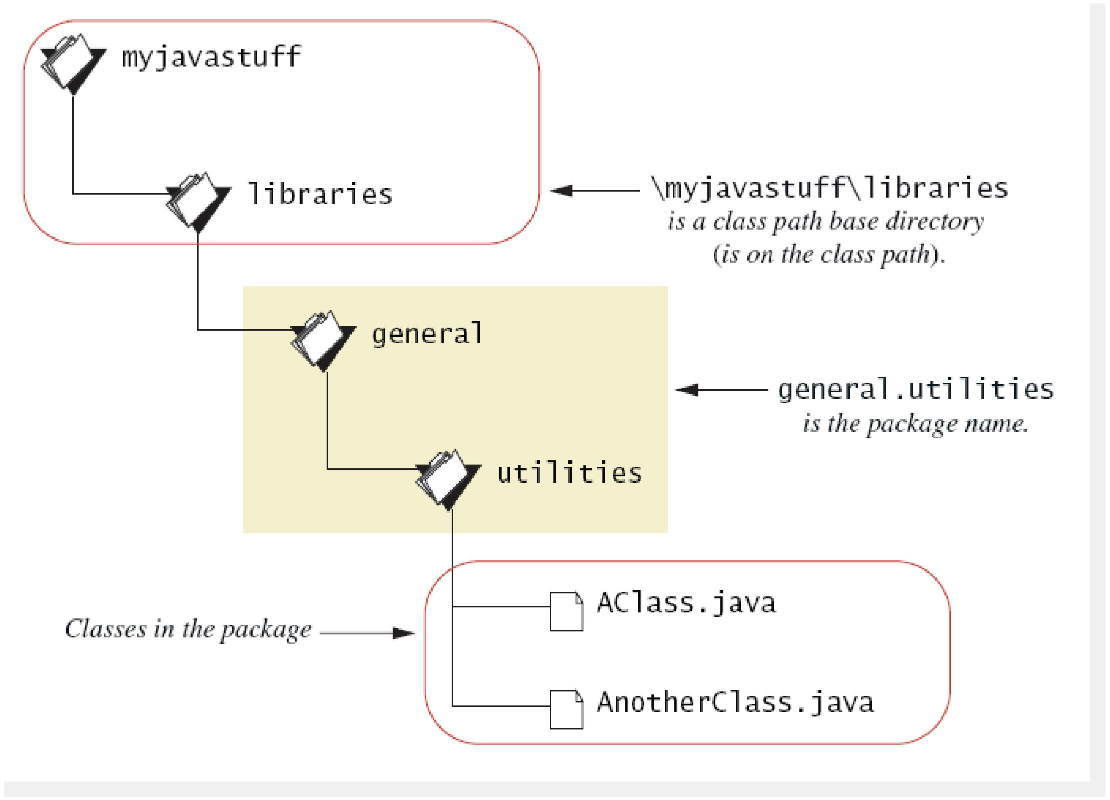
- A package is a collection of classes grouped together into a folder
- Name of folder is name of package
- Each class
    - Placed in a separate file
    - Has this line at the beginning of the file
      package `Package_Name;`
- Classes use packages by use of `import` statement

Package Names and Directories

- Package name tells compiler path name for directory containing classes of package
- Search for package begins in class path base directory
  - Package name uses dots in place of / or \
- Name of package uses relative path name starting from any directory in class path

Package Names and Directories

- Figure 6.5 A package name



Name Clashes

- Packages help in dealing with name clashes
  - When two classes have same name
- Different programmers may give same name to two classes
  - Ambiguity resolved by using the package name