10-11. More About Objects and Methods

[ECE20016/ITP20003] Java Programming

Agenda

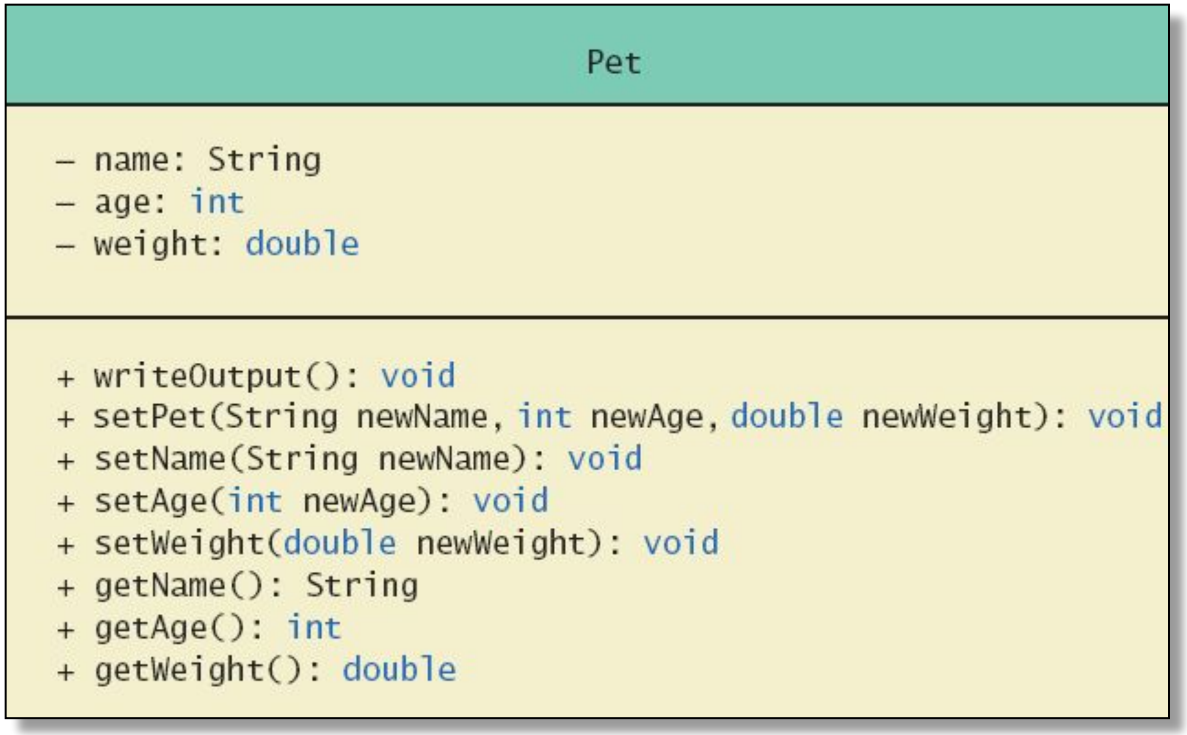
- Constructors
- Static Variables and Static Methods
- Writing Methods
- Overloading
- Enumeration As A Class
- Packages

Defining Constructors

- A special method is called when instance of an object created with *new*
  - Create instances (objects)
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions with different numbers or types of parameters

Defining Constructors

- Example class to represent pets
- Class Diagram for Class Pet



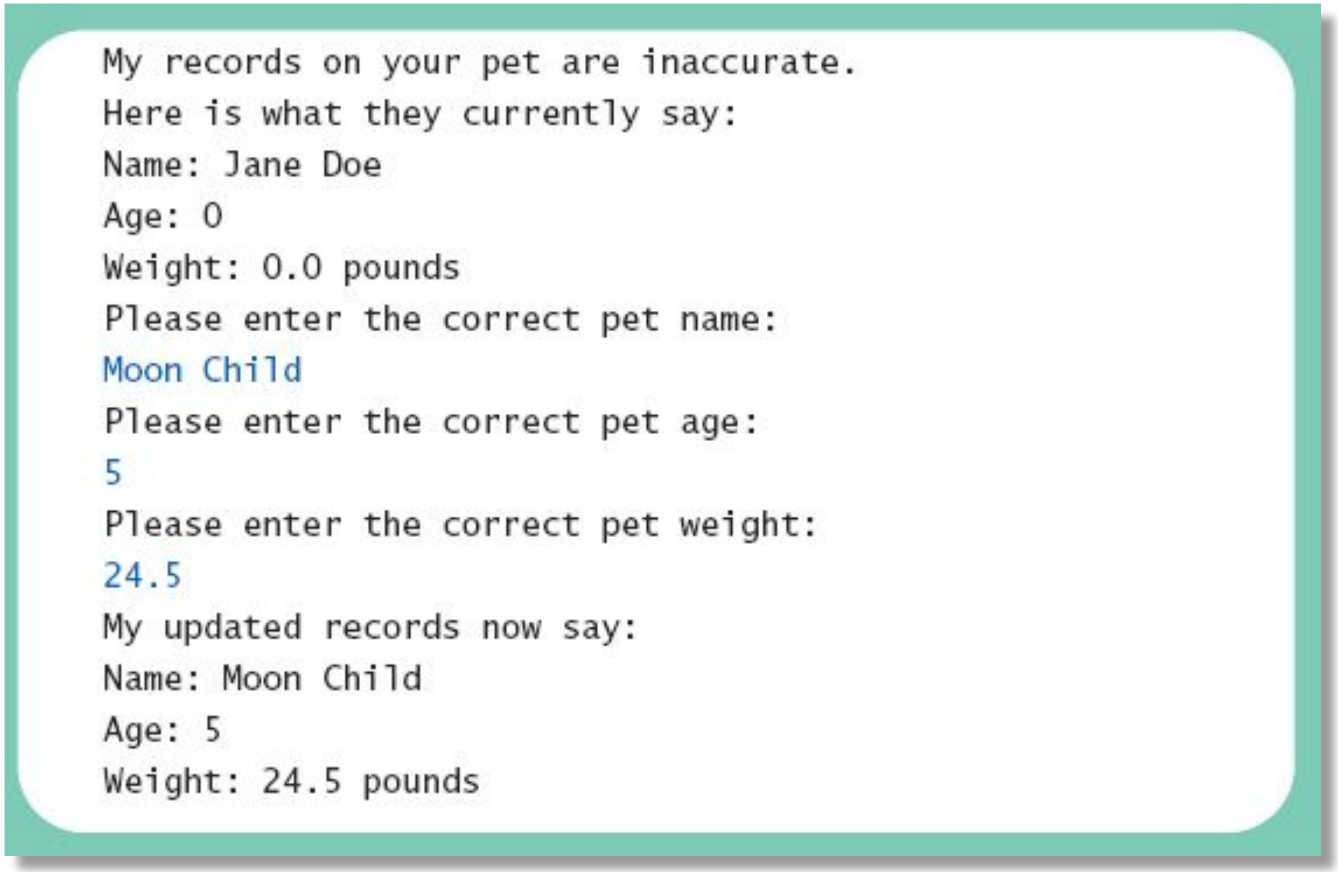
```
Pet

– name: String
– age: int
– weight: double

+ writeOutput(): void
+ setPet(String newName, int newAge, double newWeight): void
+ setName(String newName): void
+ setAge(int newAge): void
+ setWeight(double newWeight): void
+ getName(): String
+ getAge(): int
+ getWeight(): double
```

Defining Constructors

- class Pet
  - Note different constructors
    - public Pet () // default constructor
    - public Pet (String initialName, int initialAge, double initialWeight)
    - public Pet (String initialName)
    - public Pet (int initialAge)
    - public Pet (double initialWeight)

- class PetDemo
  - Pet yourPet = new Pet ("Jane Doe");

Defining Constructors



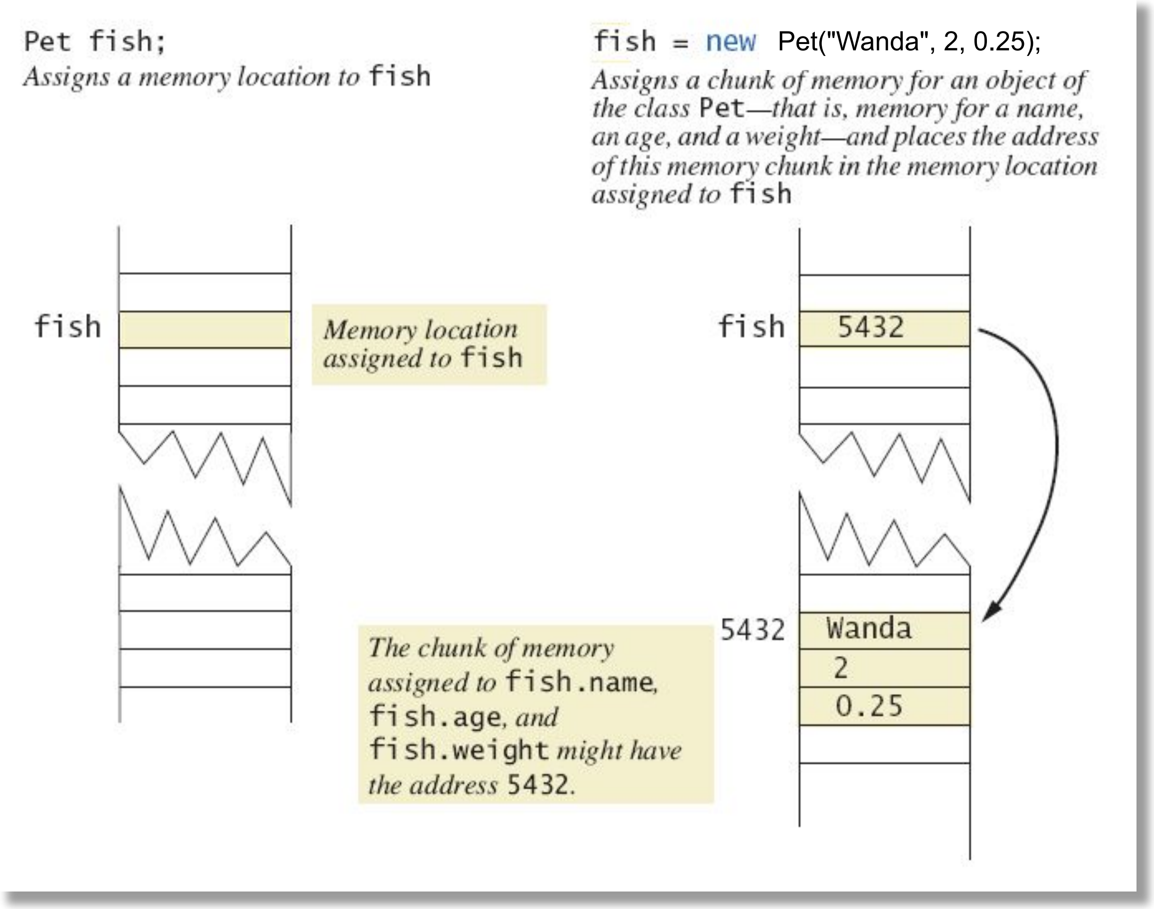
```
My records on your pet are inaccurate.
Here is what they currently say:
Name: Jane Doe
Age: 0
Weight: 0.0 pounds
Please enter the correct pet name:
Moon Child
Please enter the correct pet age:
5
Please enter the correct pet weight:
24.5
My updated records now say:
Name: Moon Child
Age: 5
Weight: 24.5 pounds
```

Defining Constructors

- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define **ANY** constructors
  - If you do define a constructor, Java will not automatically define a default constructor
- Usually default constructors not included in class diagram

Defining Constructors

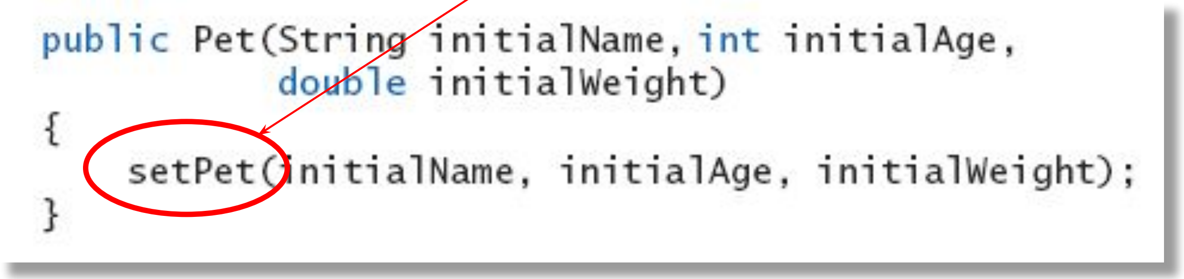
- A constructor returning a reference



Pet fish;
Assigns a memory location to fish
fish = new Pet("Wanda", 2, 0.25);
Assigns a chunk of memory for an object of the class Pet*—that is, memory for a name, an age, and a weight—and places the address of this memory chunk in the memory location assigned to* fish
fish
Memory location assigned to fish
fish
5432
5432
Wanda
2
0.25
The chunk of memory assigned to fish.name*,* fish.age*, and* fish.weight *might have the address* 5432.

Calling Methods from Other Constructors

- Constructor can call other class methods



```
public Pet(String initialName, int initialAge,
           double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

Calling Constructor from Other Constructors

- Use the *this* reference to call initial constructor

- Example

```
public Pet(String initialName, int initialAge, double initialWeight) // constructor
{
    setPet (initialName, initialAge, initialWeight);
}

public Pet(String initialName) // constructor
{
    this (initialName, 0, 0);       // calls to initial constructor
}

public void setPet(String name, int age, double weight){  // a public setter method
   this.name = name;
   this.age = age;
   this.weight = weight;
}
```

Agenda

- Constructors
- <u>**Static Variables and Static Methods**</u>
- Writing Methods
- Overloading
- Enumeration As A Class
- Packages

Static Variables

- Static variables, also called class variables, are shared by all objects of a class
  - Only one instance of the variable exists
    - Contrast with instance variables
  - Variables declared static final are considered constants – value cannot be changed
  - Variables declared static (without final) can be changed

Static Variables

```
public class StaticVarDemo {
  public int instanceVar;           // declared as public only for demonstration
  public static int staticVar;      // declared as public only for demonstration

  public static void main(String args[]){
    System.out.println("MyClass.staticVar = " + StaticVarDemo.staticVar);

    StaticVarDemo a1 = new StaticVarDemo();
    StaticVarDemo a2 = new StaticVarDemo();

    System.out.println("a1.instanceVar = " + a1.instanceVar);
    System.out.println("a2.instanceVar = " + a2.instanceVar);
    System.out.println("a1.staticVar = " + a1.staticVar);       // also possible
    System.out.println("a2.staticVar = " + a2.staticVar);       // also possible

    a1.instanceVar++;
    a1.staticVar++;
    System.out.println("a1.instanceVar = " + a1.instanceVar);
    System.out.println("a1.staticVar = " + a1.staticVar);
    System.out.println("a2.instanceVar = " + a2.instanceVar);
    System.out.println("a2.staticVar = " + a2.staticVar);
  }
}
```

Static Methods

- Some methods may have no relation to any type of object (common methods).

  Ex)
  - Compute max of two integers
  - Convert character from upper- to lower case

- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name
  - Cannot access instance variables or instance methods

Static Methods

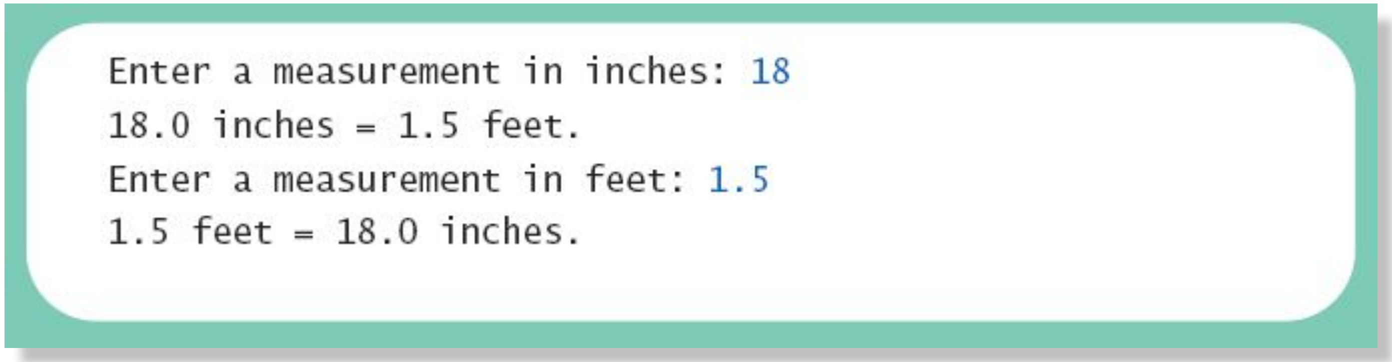
```
public class DimensionConverter
{
    public static final int INCHES_PER_FOOT = 12;

    public static double convertFeetToInches (double feet)
    {
        return feet * INCHES_PER_FOOT;
    }

    public static double convertInchesToFeet (double inches)
    {
        return inches / INCHES_PER_FOOT;
    }
}
```

Static Methods

```
import java.util.Scanner;

public class DimensionConverterDemo
{
    public static void main (String [] args)
    {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter a measurement in inches: ");
        double inches = keyboard.nextDouble ();
        double feet = DimensionConverter.convertInchesToFeet (inches);
        System.out.println (inches + " inches = " + feet + " feet.");
        System.out.print ("Enter a measurement in feet: ");
        feet = keyboard.nextDouble ();
        inches = DimensionConverter.convertFeetToInches (feet);
        System.out.println (feet + " feet = " + inches + " inches.");
    }
}
```

Static Methods

- Result



```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```

Static Methods

```
public class StaticMethodDemo {
  private int att1, att2;

  public void InstanceMethod() {
    System.out.println(
                      "This is an intance method.");
    System.out.println("\tatt1 = " + att1 + ",
                      att2 = " + att2);
  }

  public static void StaticMethod() {
    System.out.println(
                           "This is a static method.");
// static methods cannot access instance var.
//  System.out.println("\tatt1 = " + att1 + ",
//                    att2 = " + att2); // not allowed
  }
}
```

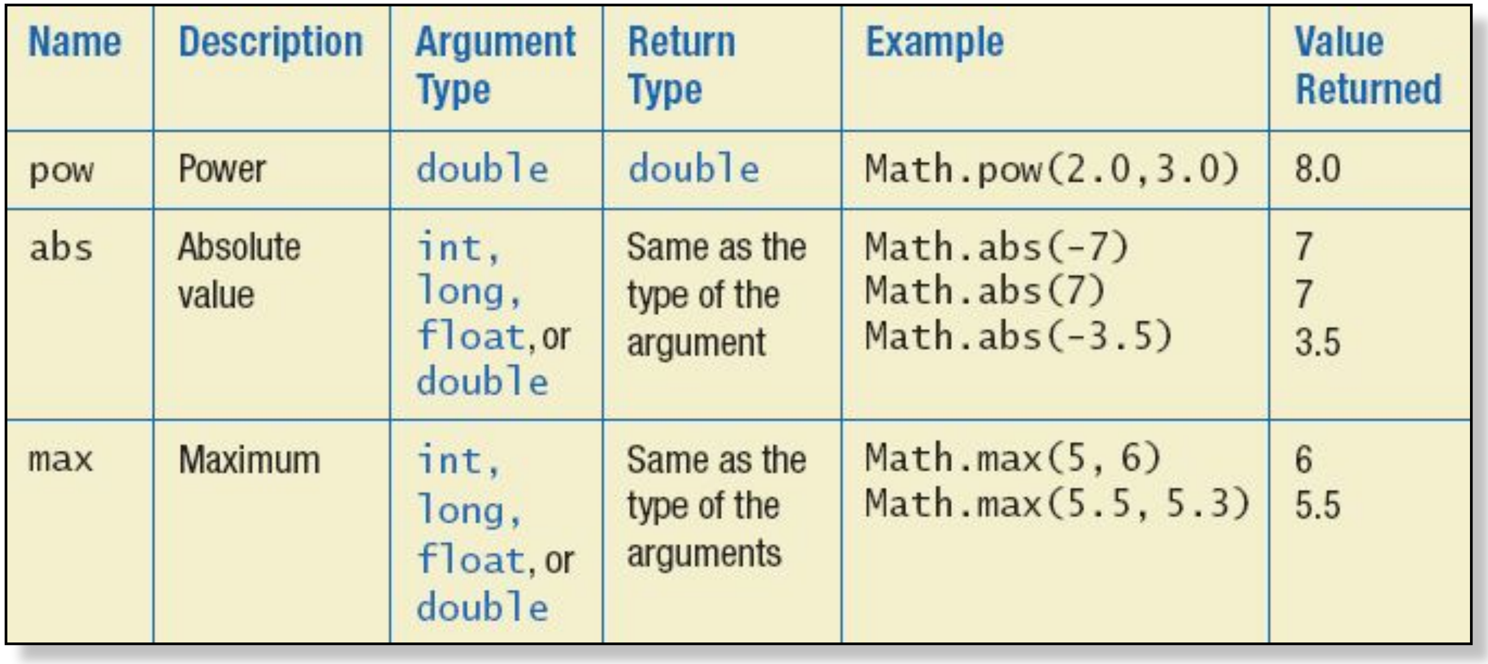
```
public class OtherClass {
 public static void main(String args[]){
   // calling instance method
  StaticMethodDemo a =
                           new StaticMethodDemo();
   a.InstanceMethod();

   // calling static method
   StaticMethodDemo.StaticMethod();
   a.StaticMethod();  // also possible
  }
}
```

Tasks of *main* in Subtasks

- Program may have complicated logic or repetitive code
- You may create static methods to accomplish subtasks
  - *main* cannot call instance methods because it's a static method.
- BUT it is recommended all actions (including logics) to be defined by methods in an object.
- Use a main method just as an entry point of your program!!!

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
  - Can scale using addition and multiplication; the following simulates rolling a six sided die

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

- Java also has a Random class to generate random numbers

Wrapper Classes

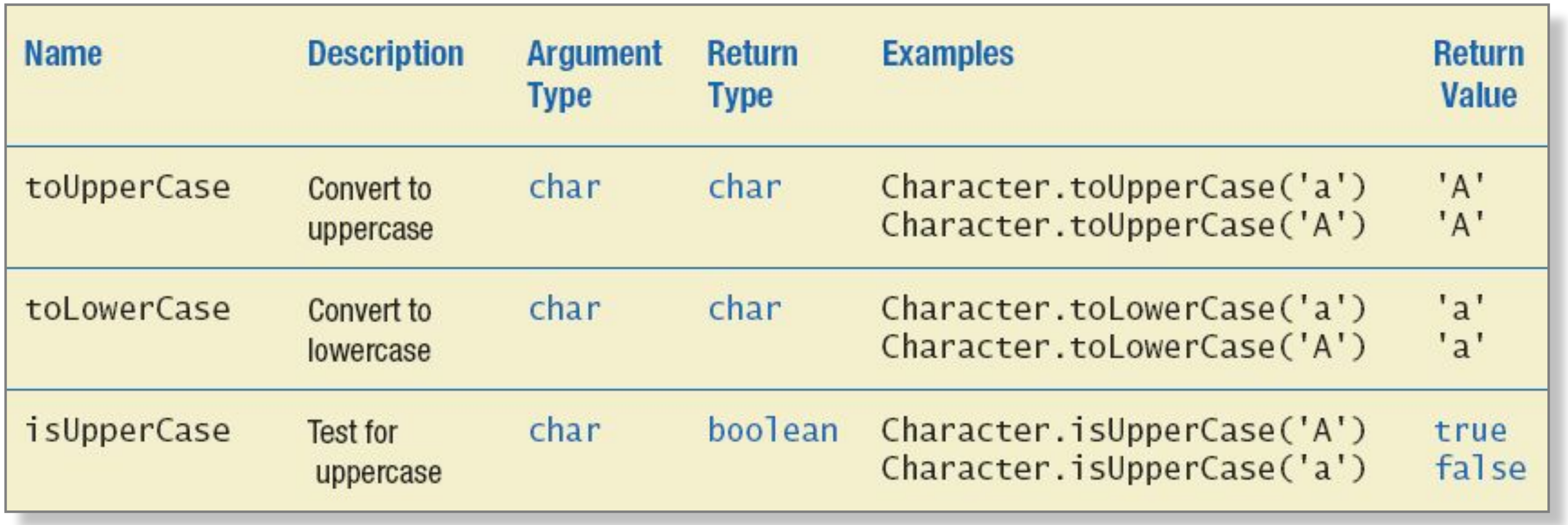
- Recall that arguments of primitive type treated differently from those of a class type
  - May need to treat primitive value as an object

- Java provides wrapper classes for each primitive type

  Byte, Short, Integer, Float, Double, Character, Boolean
  - Allow programmer to have an object that corresponds to value of primitive type
  - Contain useful predefined constants and methods
  - Wrapper classes have no default constructor
  - Wrapper classes have no *set* methods

Wrapper Classes

- Static methods in class *Character*



Name	Description	Argument Type	Return Type	Examples	Return Value

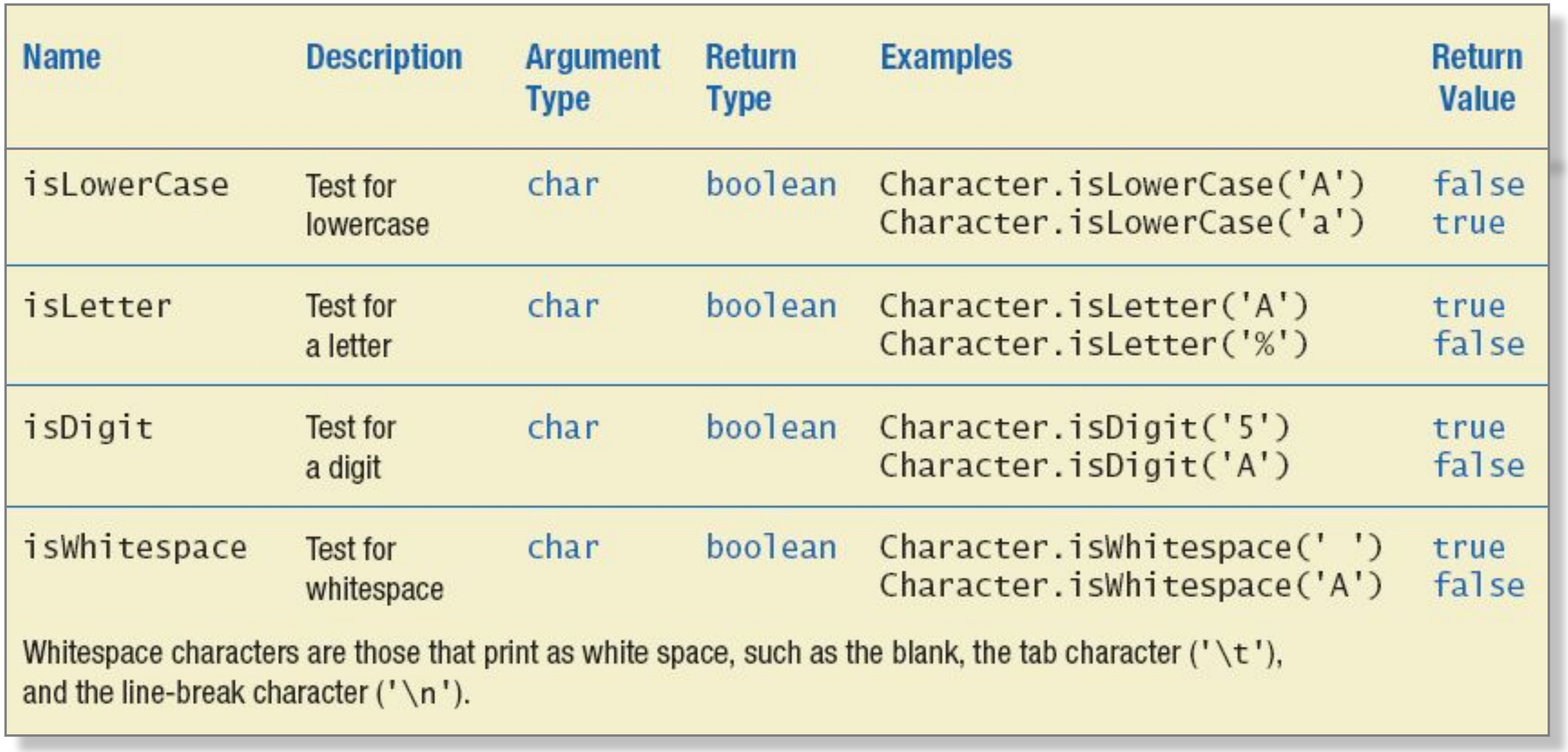
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')`
				`Character.toUpperCase('A')`
					`'A'`
					`'A'`

`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

- Static methods in class *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'`).

Agenda

- Constructors
- Static Variables and Static Methods
- **Writing Methods**
- Overloading
- Enumeration As A Class
- Packages

Formatting Output

- Algorithm to display a double amount as dollars and cents

  1. dollars = the number of whole dollars in amount.
  2. cents = the number of cents in amount. Round if there are more than two digits after the decimal point.
  3. Display a dollar sign, dollars, and a decimal point.
  4. Display cents as a two-digit integer.

DollarFormatFirstTry

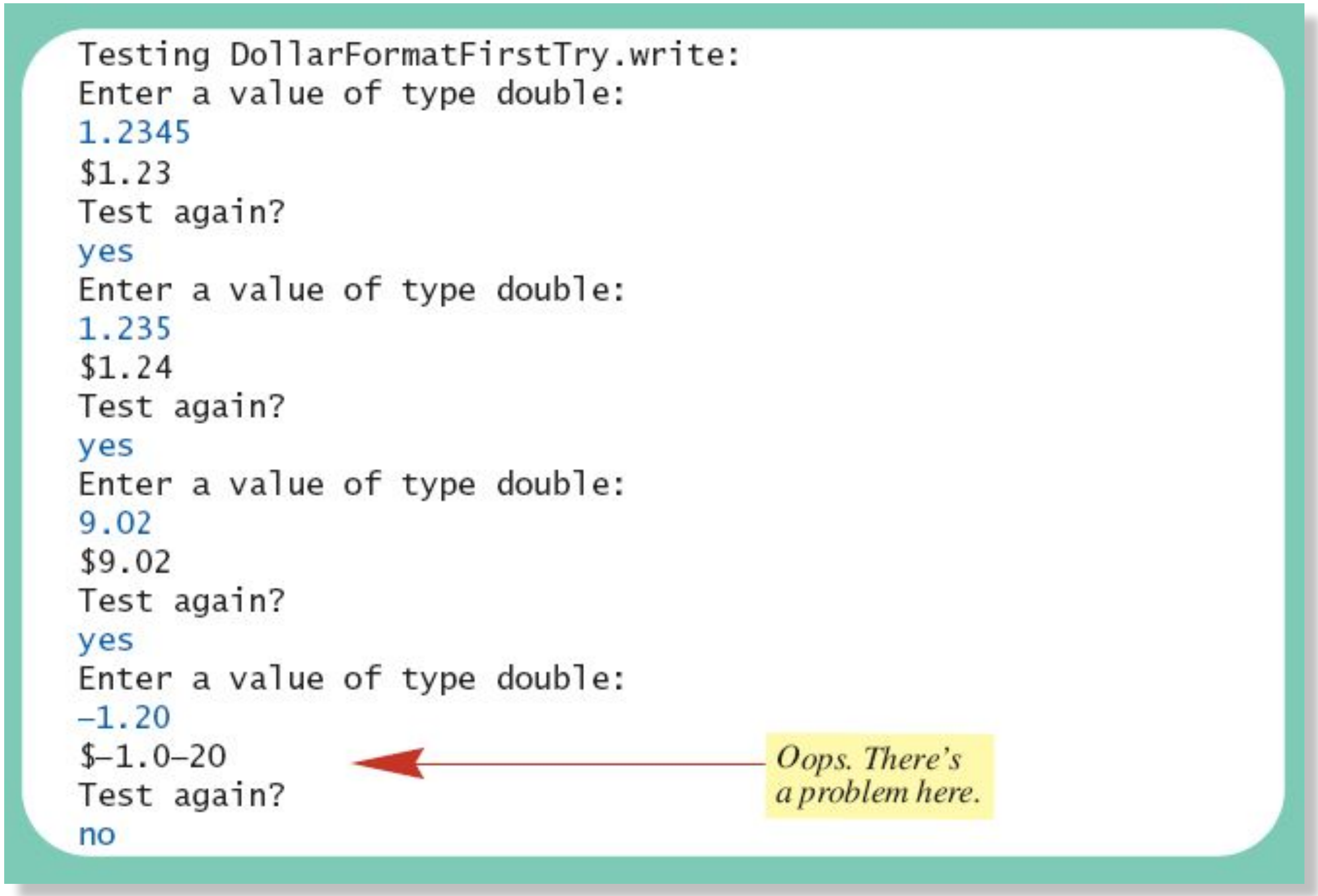
```
public class DollarFormatFirstTry
{
    public static void write (double amount)
    {
        int allCents = (int) (Math.round (amount * 100));
        int dollars = allCents / 100;
        int cents = allCents % 100;
        System.out.print ('$');
        System.out.print (dollars);
        System.out.print ('.');
        if (cents < 10) {
            System.out.print ('0');
            System.out.print (cents);
        } else
            System.out.print (cents);
    }
}
```

DollarFormatFirstTryDriver

```
import java.util.Scanner;
public class DollarFormatFirstTryDriver
{
    public static void main (String [] args)
    {
        double amount;
        String response;
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Testing DollarFormatFirstTry.write:");
        do {
            System.out.println ("Enter a value of type double:");
            amount = keyboard.nextDouble ();
            DollarFormatFirstTry.write (amount);
            System.out.println ();
            System.out.println ("Test again?");
            response = keyboard.next ();
        }
        while (response.equalsIgnoreCase ("yes"));
        System.out.println ("End of test.");
    }
}
```

Formatting Output

- Result



```
Testing DollarFormatFirstTry.write:
Enter a value of type double:
1.2345
$1.23
Test again?
yes
Enter a value of type double:
1.235
$1.24
Test again?
yes
Enter a value of type double:
9.02
$9.02
Test again?
yes
Enter a value of type double:
-1.20
$-1.0-20
Test again?
no
```

Formatting Output

- class DollarFormat
  - Note code to handle negative values
- DollarFormatFirstTryDriver will now print values correctly

Agenda

- Constructors
- Static Variables and Static Methods
- Writing Methods
- **Overloading**
- Enumeration As A Class
- Packages

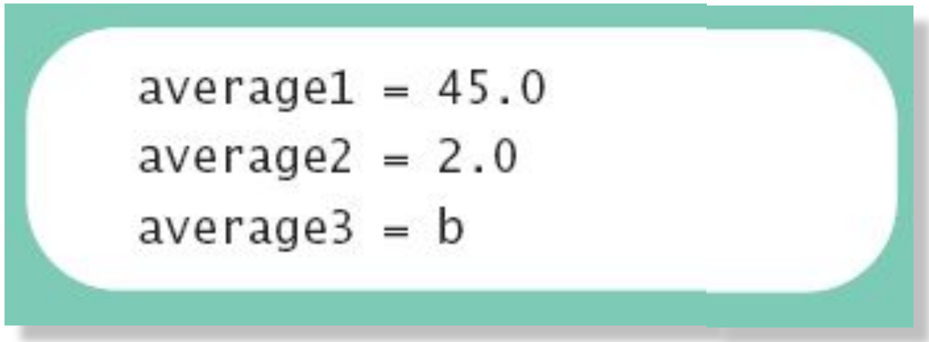
Overloading Basics

- Two or more methods may have the same name within the same class.

- 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**.
    - myMethod(int a, int b)
    - myMethod()
    - myMethod(double a, double b, double c)
    - https://en.wikipedia.org/wiki/Type_signature#Java

Overload

```
public class Overload
{
    public static void main (String [] args)
    {
        double average1 = Overload.getAverage (40.0, 50.0);
        double average2 = Overload.getAverage (1.0, 2.0, 3.0);
        char average3 = Overload.getAverage ('a', 'c');
        System.out.println ("average1 = " + average1);
        System.out.println ("average2 = " + average2);
        System.out.println ("average3 = " + average3);
    }

    public static double getAverage (double first, double second)
    {
        return (first + second) / 2.0;
    }

    public static double getAverage (double first, double second,  double third)
    {
        return (first + second + third) / 3.0;
    }

    public static char getAverage (char first, char second)
    {
        return (char) (((int) first + (int) second) / 2);
    }
}
```



```
average1 = 45.0
average2 = 2.0
average3 = b
```

Overloading and Type Conversion

- Overloading and automatic type conversion can conflict
- Remember the compiler attempts to overload before it does type conversion

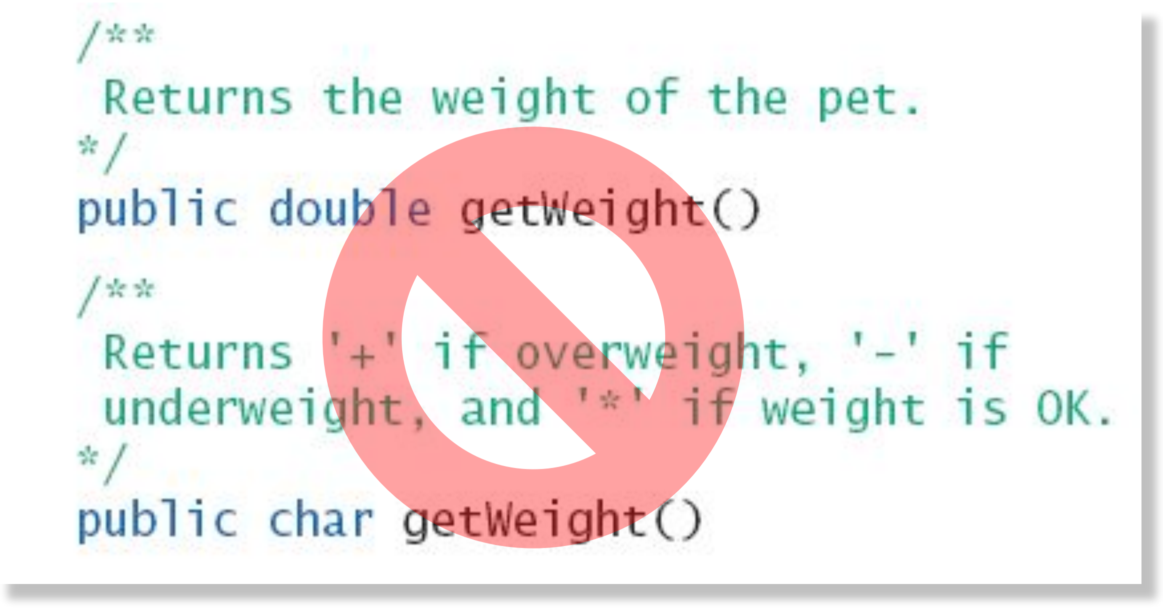
  Ex) The Pet class has two constructors

  ```
  public Pet (int initialAge)
  public Pet (double initialWeight)
  ```

  - If we pass an integer to the constructor we get the constructor for age, even if we intended the constructor for weight
- Use descriptive method names, avoid overloading

Overloading and Return Type

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



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

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

Agenda

- Constructors
- Static Variables and Static Methods
- Writing Methods
- Overloading
- **<u>Enumeration As A Class</u>**
- Packages

Enumeration as a Class

- Consider defining an enumeration for suits of cards

  enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }

- Compiler creates a class *Suit* with methods
  - equals() – tests whether current object is the same with other object
  - compareTo() - compares with other Suit object. It returns a negative, zero or a positive according to the comparison result
  - ordinal() returns the position or the ordinal value
  - toString() – returns the string form such as “HEARTS”.
  - valueOf() – eg. Suit.valueOf(“HEARTS”) returns the object Suit.HEARTS.

  For more, http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Enum.html

Enumeration as a Class

- Enhanced Enumeration *Suit*

```
enum Suit
{
    // strings as the values for the enumerated objects
    CLUBS ("black"), DIAMONDS ("red"), HEARTS ("red"),  SPADES ("black");

    private final String color;

    private Suit (String suitColor)
    {
        color = suitColor;
    }

    public String getColor ()
    {
        return color;
    }
}
```

Agenda

- Constructors
- Static Variables and Static Methods
- Writing Methods
- Overloading
- Information Hiding Revisited
- Enumeration As A Class
- **Packages**

Packages and Importing

- A package is *a named collection of classes* grouped together into a folder
  - Name of folder is name of package
  - Library of classes for use in any program
- 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

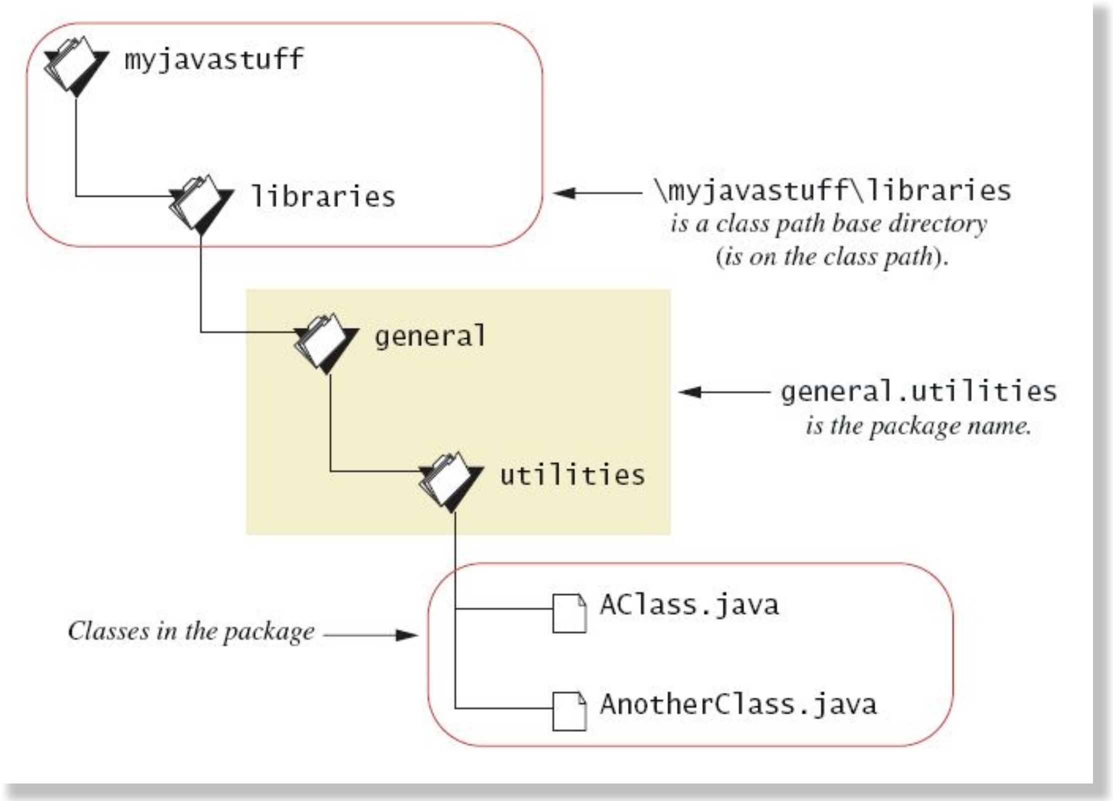
- Package name tells compiler path name for directory containing classes of package
  - Search for package begins in class path base directory
    - Package name uses dots in place of / or \

    Ex) CLASSPATH= .;C:\Program Files\Java\jre1.6.0_05\lib\;

    CLASSPATH= .; %JAVA_HOME%\lib\;
  - Name of package uses relative path name starting from any directory in class path

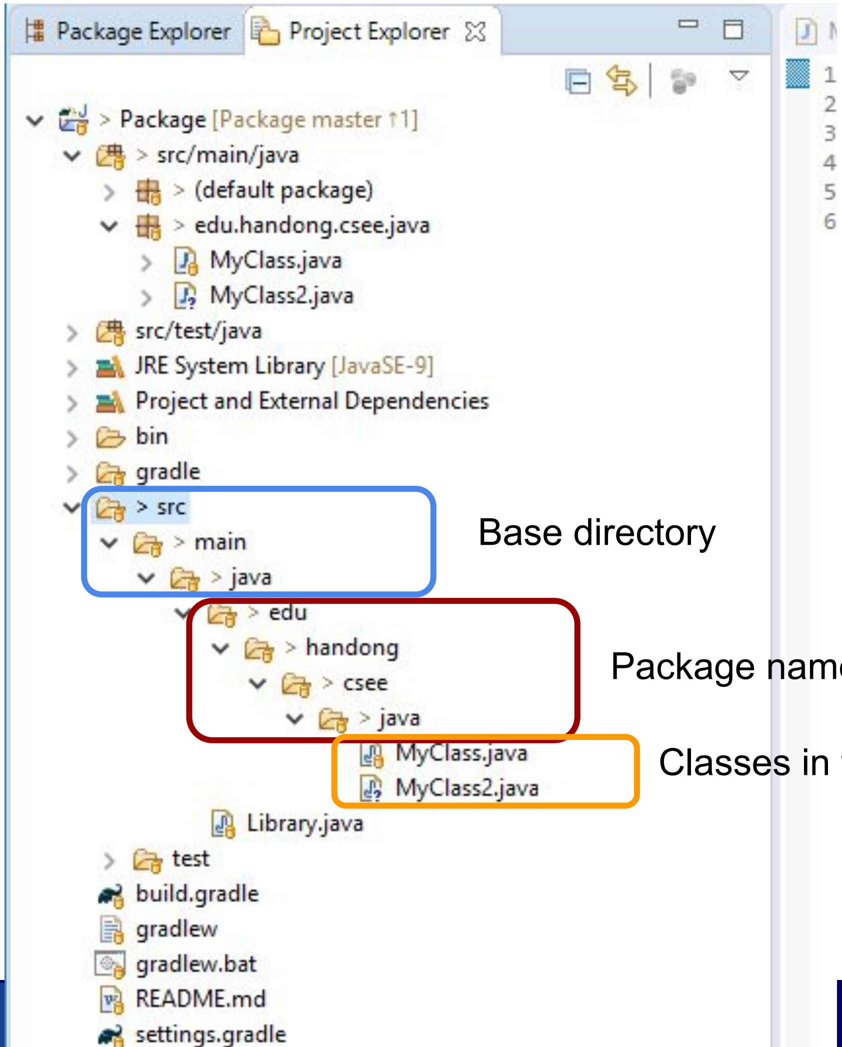
Package Names and Directories

- A package name



Package Names and Directories

- A package directories in Project Explorer in Eclipse (Window → Show View → Project Explorer)



```
Package [Package master ↑1]
  src/main/java
    (default package)
    edu.handong.csee.java
      MyClass.java
      MyClass2.java
  src/test/java
  JRE System Library [JavaSE-9]
  Project and External Dependencies
  bin
  gradle
  src
    main
      java
        edu
          handong
            csee
              java
                MyClass.java
                MyClass2.java
        Library.java
    test
  build.gradle
  gradlew
  gradlew.bat
  README.md
  settings.gradle
```

Base directory

Package name (edu.handong.csee.java)

Classes in the package

Name Clashes (conflicts)

- 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