컴퓨터프로그래밍II

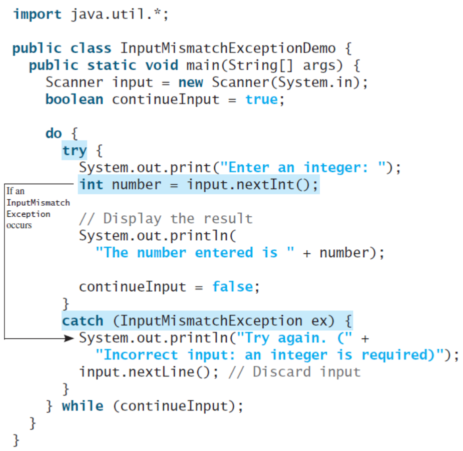
Jong-Kyou Kim, PhD

2016-11-11

Review: Exception

```
public class MyException {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    for(;;) {
      try {
        ...
      }
      catch (InputMismatchException ex) {
        ...
      }
      catch (ArithmeticException ex) {
        ...
      }
      finally {
        ...
      }
    }
  }
}
```

Exception


If an InputMismatch Exception occurs

```java
import java.util.*;

public class InputMismatchExceptionDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean continueInput = true;

    do {
      try {
        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        // Display the result
        System.out.println(
          "The number entered is " + number);

        continueInput = false;
      }
      catch (InputMismatchException ex) {
        System.out.println("Try again. (" +
          "Incorrect input: an integer is required)");
        input.nextLine(); // Discard input
      }
    } while (continueInput);
  }
}
```

그림: am

Exception

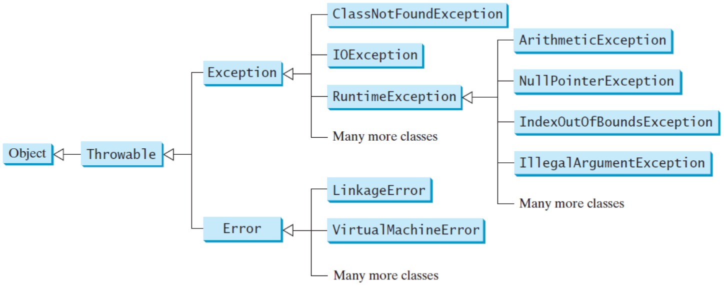


그림: an

Review: Object oriented programming (OOP)

```
+----------------------------------+
| LinearEquation                   |
| ------------------------------   |
| - a,b,c,d,e,f: double            |
| ------------------------------   |
| + LinearEquation()               |
| + getA(),getB(),....: double     |
| + isSolvable(): boolean          |
| + getX(): dobule                 |
| + getY(): double                 |
+----------------------------------+
```

Jong-Kyou Kim, PhD

Review: Object oriented programming (OOP)

```
public class LinearEquation {
  private double a, b, c, d, e, f;
  LinearEquation(double a, double b, double c,
                 double d, double e, double f) {
    this.a = a;
    this.b = b;
    this.c = c;
    this.d = d;
    this.e = e;
    this.f = f;
  }
  ...
  }
```

Review: Object oriented programming (OOP)

```
public class LinearEquation {
  ...
  double getA() { return a; }
  double getB() { return b; }
  double getC() { return c; }
  double getD() { return d; }
  double getE() { return e; }
  double getF() { return f; }
  ...
}
```

Review: Object oriented programming (OOP)

```
public class LinearEquation {
  ...
  private double getDisc() {
    return a*d - b*c;
  }
  ...
}
```

Review: Object oriented programming (OOP)

```
public class LinearEquation {
  ...
  boolean isSolvable() {
    return getDisc() != 0.0;
  }
  double getX() {
    return (e*d - b*f)/getDisc();
  }
  double getY() {
    return (a*f - e*c)/getDisc();
  }
}
```

Visibility

- What's the difference?

```
public class LinearEquation {
  boolean isSolvable() {
    return getDisc() != 0.0;
  }
}
```

```
public class LinearEquation {
  public boolean isSolvable() {
    return getDisc() != 0.0;
  }
}
```

$\longrightarrow$ Package private

Source files

```
work/Test.java
work/C.java
```

Useful class

```
public class C {
  int m() {
    return 0;
  }
  static int s() {
    return 0;
  }
}
```

Test program

```
public class Test {
  public static void main(String [] args) {
    C x = new C();
    System.out.println(x.m());
    System.out.println(x.s());
  }
}
```

Directory

```
work/Test.java
work/p0/C.java
```

```
javac Test.java
Test.java:3: error: cannot find symbol
    C x = new C();
    ^
  symbol:   class C
  location: class Test
```

Importing from package

```
import p0.C;

public class Test {
  public static void main(String [] args) {
    C x = new C();
    System.out.println(x.m());
    System.out.println(x.s());
  }
}
```

$\longrightarrow$ Still error!

Package declaration

```
package p0;

public class C {
  int m() {
    return 0;
  }
  static int s() {
    return 0;
  }
}
```

$\longrightarrow$ Still error!

Package private

```
package p0;

public class C {
  public int m() {
    return 0;
  }
  public static int s() {
    return 0;
  }
}
```

Package and visibility

- Classes are grouped in a package
- A class in a package can access any other class in the package
- A class in a package can access other classes in a package
  - But it has to be declared as `import`
  - Each class shoud notify where it belongs by using `package`
- Any class, method, or data field should be declared `public` to be used in a class declared in a different package

Abstraction

- Simplifies complex structure, concept, operations
- Expose only contracts (methods, constructors, data fields)

  $\longrightarrow$ UML
- Detailed implemantations are hidden under implementation

  $\longrightarrow$ source code

Abstraction: concept

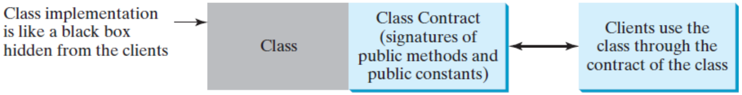


그림: a

Abstraction: example

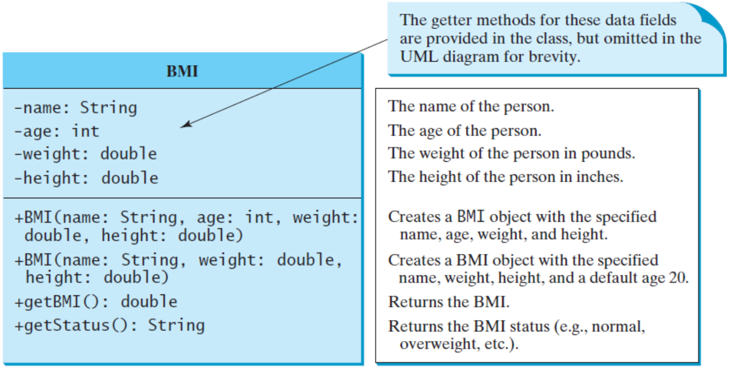

The getter methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
BMI
-name: String
-age: int
-weight: double
-height: double
+BMI(name: String, age: int, weight: double, height: double)
+BMI(name: String, weight: double, height: double)
+getBMI(): double
+getStatus(): String
The name of the person.
The age of the person.
The weight of the person in pounds.
The height of the person in inches.
Creates a BMI object with the specified name, age, weight, and height.
Creates a BMI object with the specified name, weight, height, and a default age 20.
Returns the BMI.
Returns the BMI status (e.g., normal, overweight, etc.).

그림: b

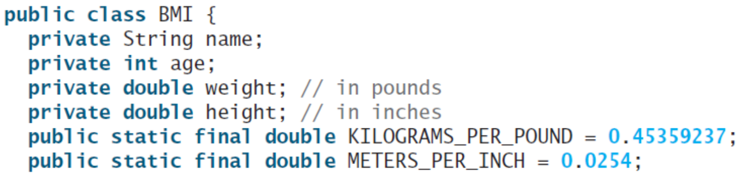
Contract

```java
public class UseBMIClass {
  public static void main(String[] args) {
    BMI bmi1 = new BMI("Kim Yang", 18, 145, 70);
    System.out.println("The BMI for " + bmi1.getName() + " is "
      + bmi1.getBMI() + " " + bmi1.getStatus());

    BMI bmi2 = new BMI("Susan King", 215, 70);
    System.out.println("The BMI for " + bmi2.getName() + " is "
      + bmi2.getBMI() + " " + bmi2.getStatus());
  }
}
```

```
The BMI for Kim Yang is 20.81 Normal
The BMI for Susan King is 30.85 Obese
```

그림: c

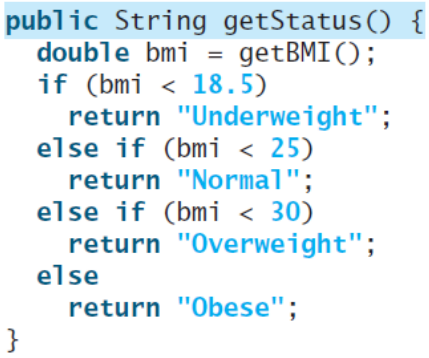
Implementation



```
public class BMI {
  private String name;
  private int age;
  private double weight; // in pounds
  private double height; // in inches
  public static final double KILOGRAMS_PER_POUND = 0.45359237;
  public static final double METERS_PER_INCH = 0.0254;
```

그림: d

Implementation



```
public String getStatus() {
  double bmi = getBMI();
  if (bmi < 18.5)
    return "Underweight";
  else if (bmi < 25)
    return "Normal";
  else if (bmi < 30)
    return "Overweight";
  else
    return "Obese";
}
```

그림: e

Class relations

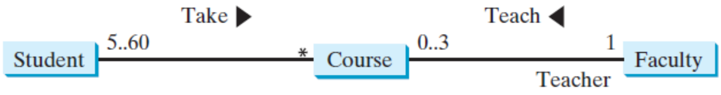
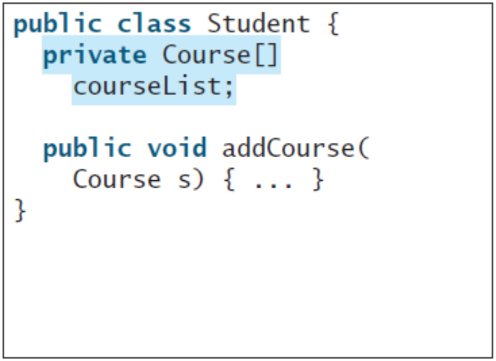


그림: f

- A student take a course, a faculty teaches a course
  - noun: object (abstracted by a class)
  - verb: relation

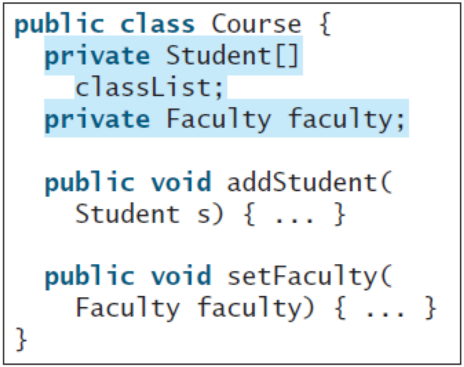
Implemnation of class relation



```
public class Student {
  private Course[]
    courseList;

  public void addCourse(
    Course s) { ... }
}
```

그림: g

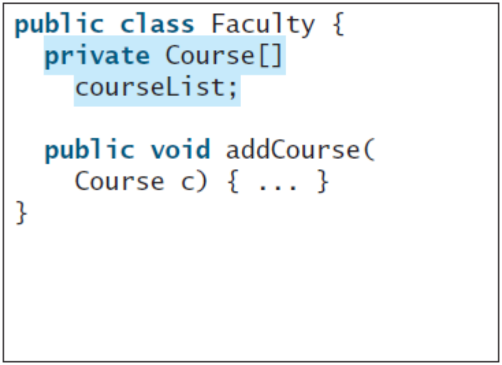
Implemnation of class relation



```
public class Course {
  private Student[]
    classList;
  private Faculty faculty;

  public void addStudent(
    Student s) { ... }

  public void setFaculty(
    Faculty faculty) { ... }
}
```

그림: h

Implemnation of class relation



```
public class Faculty {
  private Course[]
    courseList;

  public void addCourse(
    Course c) { ... }
}
```

그림: i

Has-a: special relation

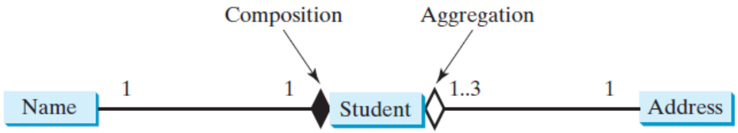
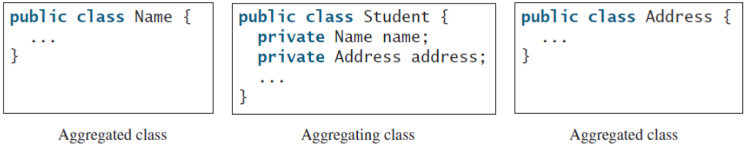


그림: j

- Exclusive: filled-diamond
- Shared: open-diamond

Jong-Kyou Kim, PhD

Implementation of has-a



```
public class Name {
  ...
}
```

Aggregated class

```
public class Student {
  private Name name;
  private Address address;
  ...
}
```

Aggregating class

```
public class Address {
  ...
}
```

Aggregated class

그림: k

Recursive relation

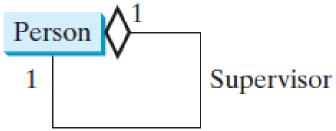
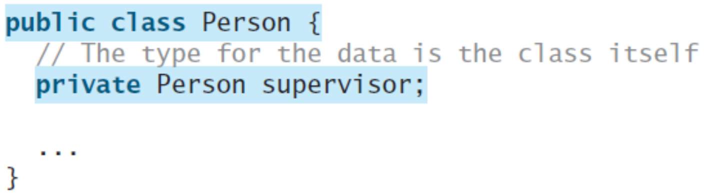


그림: l

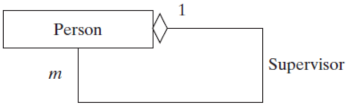
Implementation of recursive relation



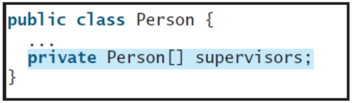
```
public class Person {
  // The type for the data is the class itself
  private Person supervisor;

  ...
}
```

그림: m

Multiple relation



(a)



```
public class Person {
  ...
  private Person[] supervisors;
}
```

(b)

그림: n

Wrap-up

- Exception handling simplifies programming
- Java program is composed of a huge number of classes
- The keyword `public` controls the visibility among packages
- Object-oriented programming is useful in mananging complexity (abstraction)
- Understanding the relationship among classes is the key to learning OOP