


# Computer Language



Java Basic Packages

# Agenda

---

- Exception Handling
- Enumeration
- Java Packages – Part I

# Exception Handling

Enumeration  
Java Packages

# Exception

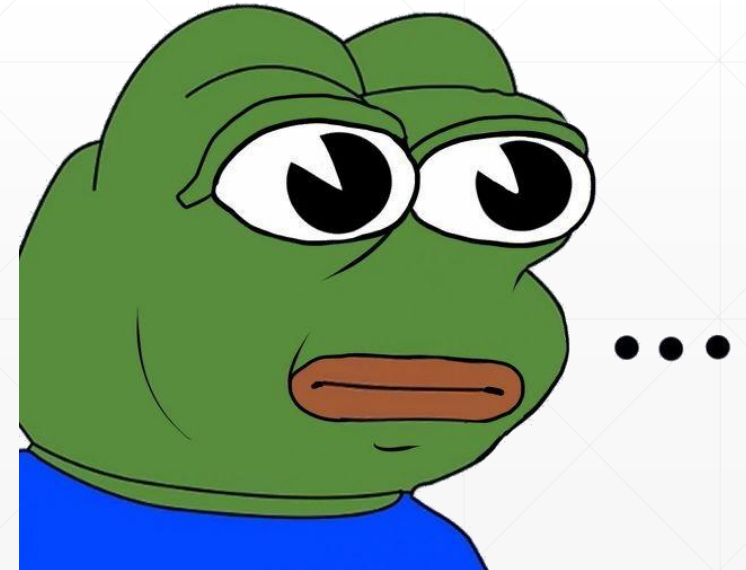
---

- Event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions
  - Shorthand for the phrase “exceptional event”
  - Generally related with “error”
    - Invalid manipulation of a program by the user
    - Developer’s incorrect logics
    - ...
- What happens if exception occurs?
  - Your program will be crashed
  - Before crashing, your program can handle the exceptions!

# Exception (cont'd)

## ■ When exception occurs?

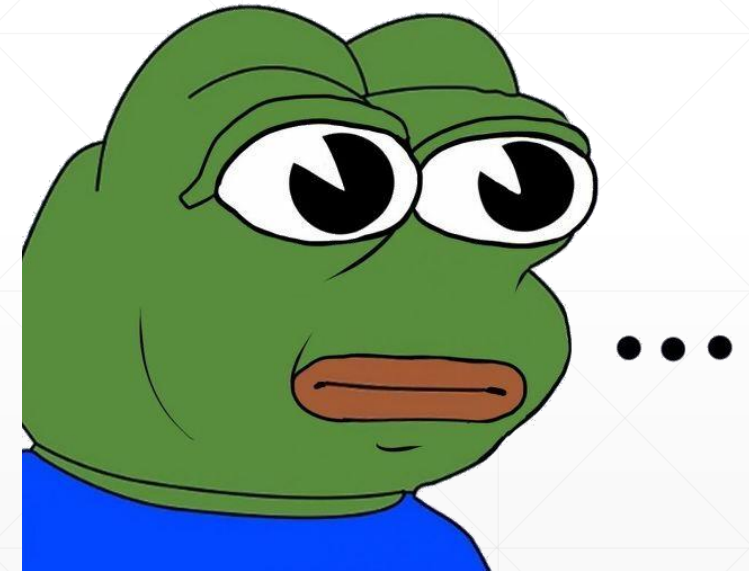
- Dividing an integer by zero
- Accessing an element of an array with an index greater than the length of the array
- Reading a file that does not exist
- Entering a string value to the position where an integer value is required.
- ...



# Exception (cont'd)

## ■ What kind of exception occurs?

Exception class	When?
ArithmeticException	Dividing an integer by zero
NullPointerException	Referencing a null reference
ClassCastException	Casting to the invalid type
OutOfMemoryError	Not enough memory
ArrayIndexOutOfBoundsException	Accessing an invalid index of an array
IllegalArgumentException	Passing invalid arguments
IOException	IO operation failure
NumberFormatException	Invalid number conversion
InputMismatchException	Invalid use of Scanner methods

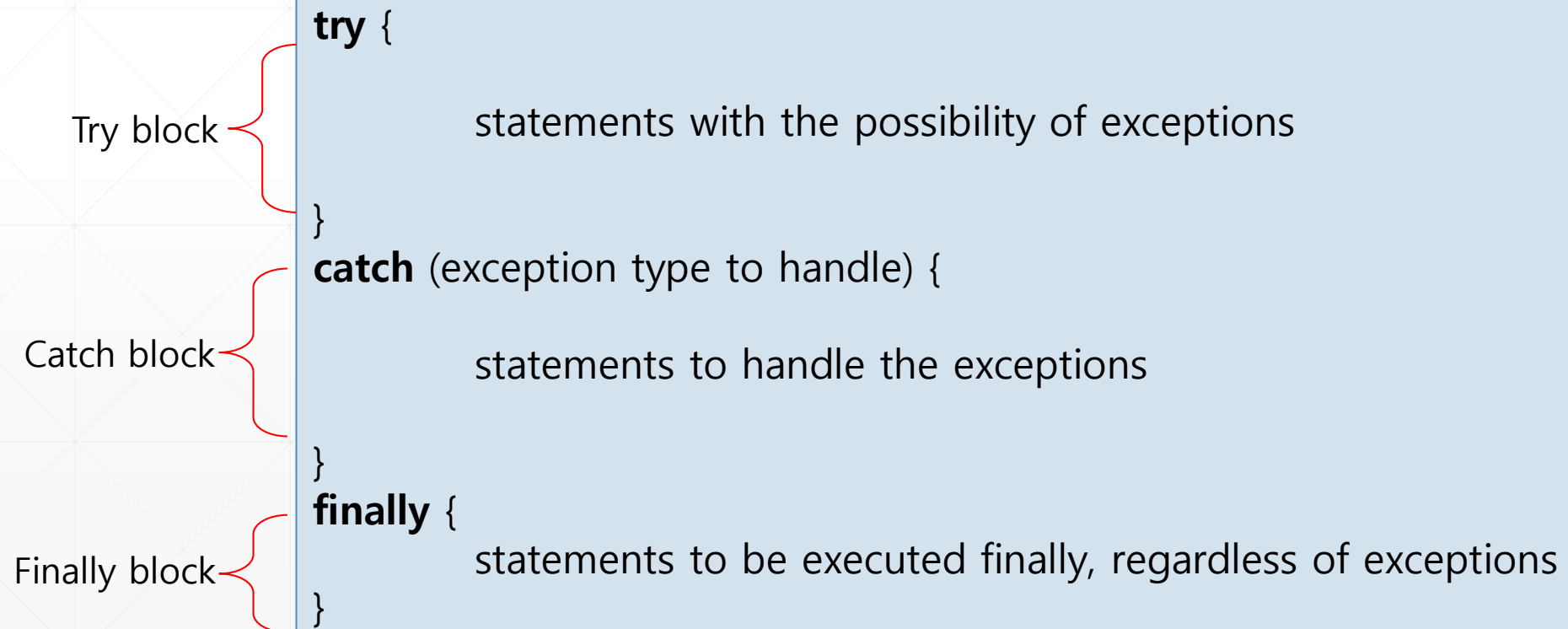


“Exception” class is a superclass of all other specific exception classes!

# Exception: Try-Catch-Finally (cont'd)

## ■ So, how to handle exceptions?

- Use Try-Catch(-Finally) statement!
  - Finally block can be omitted

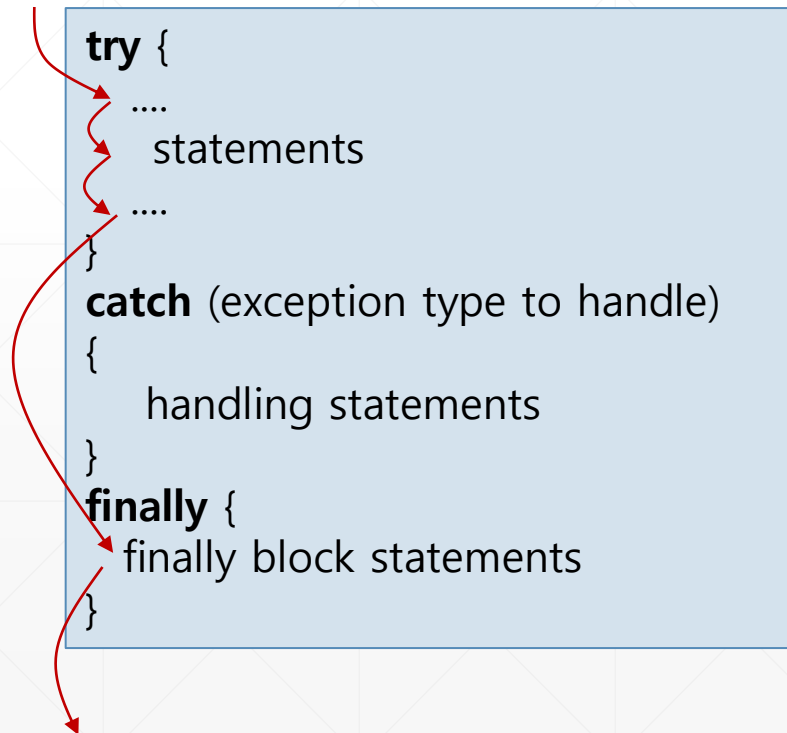


# Exception: Try-Catch-Finally (cont'd)

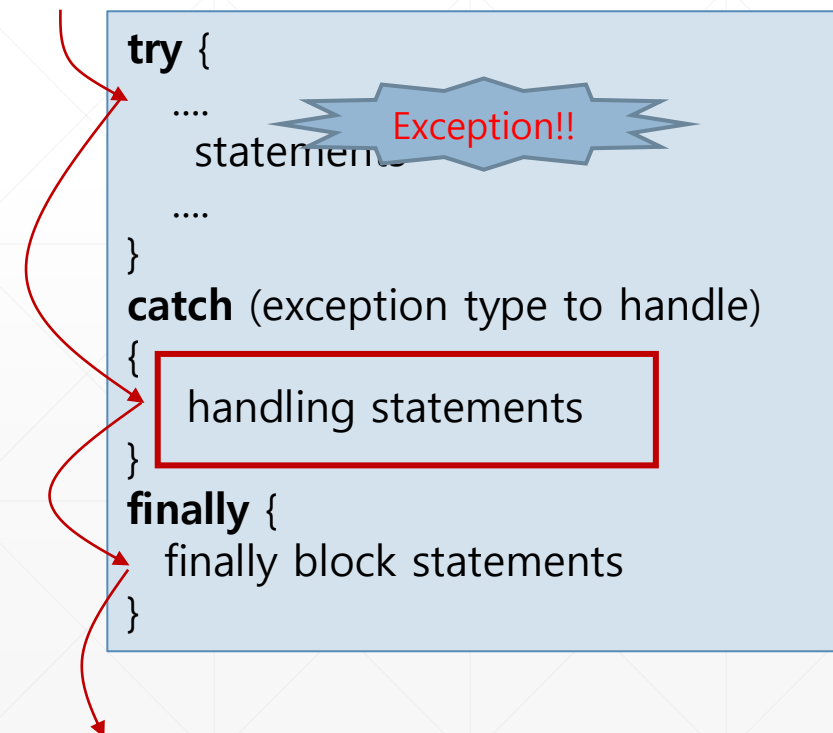
## ■ So, how to handle exceptions?

- Use Try-Catch(-Finally) statement!

Normal case that no exceptions occur in the try block



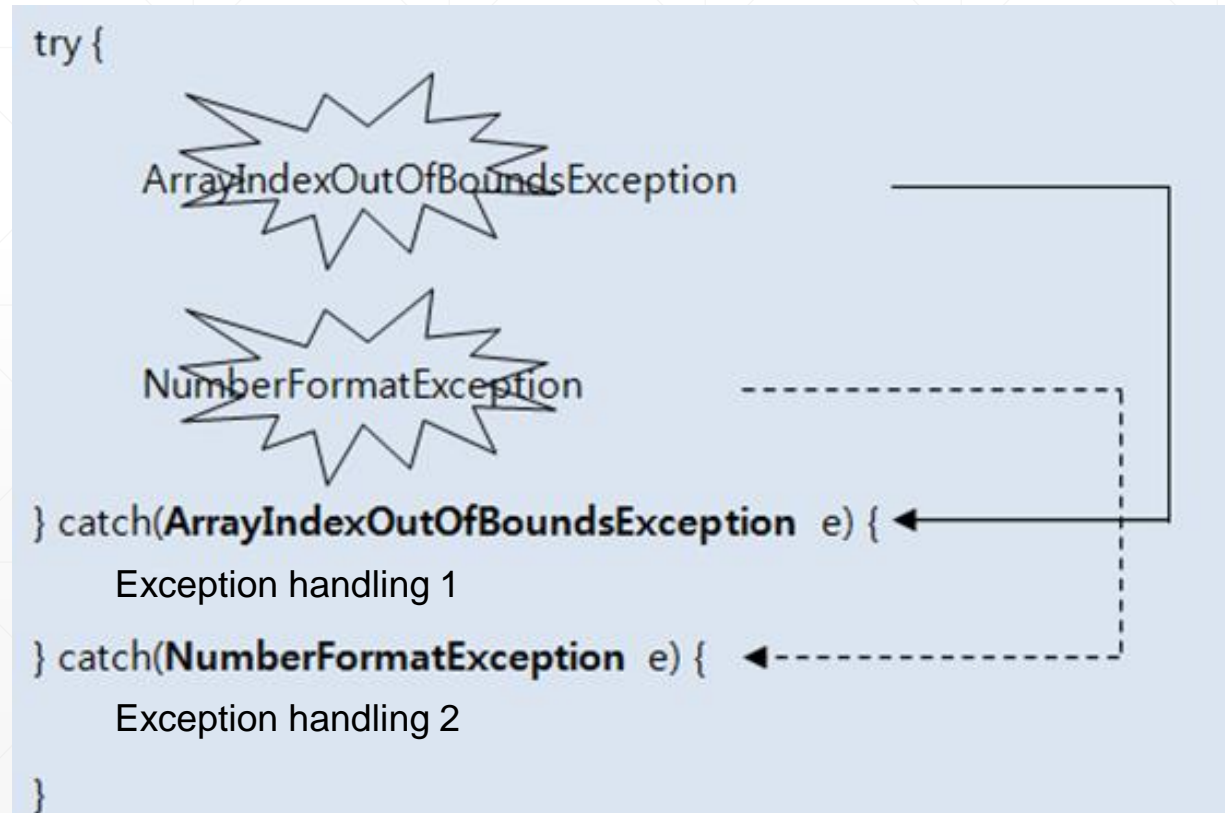
Error case that an exception occurs in the try block





# Exception: Try-Catch-Finally (cont'd)

- So, how to handle exceptions?
  - Use Try-Catch(-Finally) statement!
  - Multiple catch statements are also allowed



# Exception: Examples

## ■ ArithmeticException

```
import java.util.Scanner;

public class DivideByZero {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int dividend;
        int divisor;

        System.out.print("Input your number:");
        dividend = scanner.nextInt();
        System.out.print("Input your divisor:");
        divisor = scanner.nextInt();
        System.out.println(dividend + " divided by " + divisor + " is " + dividend/divisor );
        scanner.close();
    }
}
```

Exception occurs  
when divisor is 0

# Exception: Examples (cont'd)

## ■ ArrayIndexOutOfBoundsException

```
public class ArrayException {  
    public static void main (String[] args) {  
        int[] intArray = new int[5];  
        intArray[0] = 0;  
        try {  
            for (int i=0; i<5; i++) {  
                intArray[i+1] = i+1 + intArray[i];  
                System.out.println("intArray["+i+"]"+"="+intArray[i]);  
            }  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("out of index!");  
        }  
    }  
}
```

Exception occurs  
when i is 4

# Exception: Examples (cont'd)

## ■ NumberFormatException

```
public class NumException {  
    public static void main (String[] args) {  
        String[] stringNumber = {"23", "12", "3.141592", "998"};  
        String test = null;  
        int i=0;  
        try {  
            for (i=0; i<stringNumber.length; i++) {  
                int j = Integer.parseInt(stringNumber[i]);  
                System.out.println("The value after converting to integer number is " + j);  
                if(i % 2 == 1) System.out.println(test.length());  
            }  
        }  
        catch (NumberFormatException e) {  
            System.out.println(stringNumber[i] + " cannot be converted to integer number.");  
        }  
        catch (NullPointerException e){  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

Exception occurs when  
converting "3.141592"

Exception occurs when  
i is an odd-number

# Exception: Error Information

## ■ getMessage()

- Can take the error message for the exception
- Used in the catch block

## ■ printStackTrace()

- Print all the history of tracing the exception source to the console

```
try {
```



```
} catch(Exception type e) {  
    // take the message of the exception  
    String message = e.getMessage();  
  
    // trace the path of exception  
    e.printStackTrace();  
}
```

# Exception: Debugging

## ■ Breakpoint using IDE



```
NumException.java x
1 public class NumException {
2     public static void main (String[] args) {
3         String[] stringNumber = {"23", "12", "3.141592", "998"};
4         String test = null;
5         int i=0;
6         try {
7             for (i=0; i<stringNumber.length; i++) {
8                 int j = Integer.parseInt(stringNumber[i]);
9                 System.out.println("The value after converting to integer number is " + j);
10                //if(i % 2 == 1) System.out.println(test.length());
11            }
12        }
13        catch (NumberFormatException e) {
14            System.out.println(stringNumber[i] + " cannot be converted to integer number.");
15        }
16        catch (NullPointerException e){
17            System.out.println(e.getMessage());
18        }
19    }
20 }
21 }
```

# Exception: Debugging (cont'd)

## ■ Breakpoint using IDE

```
6      try {
7          for (i=0; i<stringNumber.length; i++) {
8              int j = Integer.parseInt(stringNumber[i]);  stringNumber: ["23", "12", "3.141592", "998"]
9              System.out.println("The value after converting to integer number is " + j);
10             //if(i % 2 == 1) System.out.println(test.length());
11         }
12     }
13     catch (NumberFormatException e) {
14         System.out.println(stringNumber[i] + " cannot be converted to integer number.");
15     }
16     catch (NullPointerException e){
17         System.out.println(e.getMessage());
18     }
19
20 }
21
22 }
```

Debug: NumException

Debugger Console

Frames

- main:8, NumException

Variables

- args = {String[0]@801} []
- stringNumber = {String[4]@802} ["23", "12", "3.141592", "998"]
- test = null
- i = 0
- stringNumber[i] = "23"
- stringNumber.length = 4

← Current status of variables



Exception Handling

# Enumeration

Java Packages



# Enumeration: Review

## ■ Enumeration

- Special data type to store a set of constants
- Common example
  - Representing compass directions: {NORTH, SOUTH, EAST, WEST}
  - Representing the days of a week: {SUNDAY, MONDAY, TUESDAY, ..., SATURDAY}
- Enum-type variable must be equal to one of the values that have been predefined for it
- Declaration 

public enum Enumtype { ...(a set of enum constants) }

  - Need to be declared in the java file with the same Enumtype name
  - Enum constant should be CAPITAL (naming convention)

```
public enum Week { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, ... }
```

```
public enum LoginResult { LOGIN_SUCCESS, LOGIN_FAILED }
```

# Enumeration: Review (cont'd)

## ■ Enumeration

- Declaration of Enum type variable

```
Enumtype variableName;
```

```
Week today;
```

```
Week reservationDay;
```

- Assigning a value to Enum type variable

- Value must be equal to one of the values that have been predefined for it

```
Enumtype variableName = Enumtype.constant;
```

```
Week today = Week.SUNDAY;
```

- Enum type is a kind of reference type

- Enum-type variable can use null literal

```
Week birthday = null;
```

# Enumeration: Review (cont'd)

## ■ Example)

Hello.java

```
public class Hello {  
    public static void main(String[] args) {  
  
        Weekday myDay = Weekday.FRIDAY;  
  
        switch (myDay) {  
            case MONDAY:  
                System.out.println("Mondays are bad.");  
                break;  
            case FRIDAY:  
                System.out.println("Fridays are better.");  
                break;  
            case SATURDAY: case SUNDAY:  
                System.out.println("Weekends are best.");  
                break;  
            default:  
                System.out.println("Midweek days are so-so.");  
                break;  
        }  
    }  
}
```

Weekday.java

```
public enum Weekday {  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY,  
    SUNDAY  
}
```

# Enumeration: Class-usage

## ■ Enumeration is actually a kind of Java class

- So, we can use various features of Java class!
- Fields, constructor, methods, etc

## ■ Constructor

- Access modifier: private
  - We cannot create an enum object explicitly
- Called for each constant definition
  - At the time of enum class loading!
- 'this' keyword refers to the created constant itself

```
public enum Weekday {  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY,  
    SUNDAY;  
  
    Weekday(){  
        System.out.println(this + "was called!");  
    }  
}
```

# Enumeration: Class-usage (cont'd)

## ■ Custom fields

- We can assign some custom values to the constant
  - Syntax: `CONSTANT(...values...)`
- Field definition for a custom value required
- Constructor with arguments required
  - To assign custom values to the field

- If enum class has fields/methods, then the constant definitions must end with a semicolon

```
public enum Weekday {  
    MONDAY("NO"),  
    TUESDAY("No"),  
    WEDNESDAY("no"),  
    THURSDAY("yes"),  
    FRIDAY("YES!"),  
    SATURDAY("YEAH~!"),  
    SUNDAY("SAD...");  
  
    public String text; // field for custom message  
    Weekday(String text){  
        System.out.println(this + " was called!");  
        this.text = text; // assigning custom messages  
    }  
}
```

# Enumeration: Class-usage (cont'd)

## ■ Custom fields

```
public class Hello {  
    public static void main(String[] args) {  
  
        Weekday myDay = Weekday.SATURDAY;  
  
        switch (myDay) {  
            case MONDAY:  
            case FRIDAY:  
            case SATURDAY:  
            case SUNDAY:  
                System.out.println(myDay.name()+ " is " + myDay.text);  
                break;  
            default:  
                System.out.println("Midweek days are so-so.");  
                break;  
        }  
    }  
}
```

```
public enum Weekday {  
    MONDAY("NO"),  
    TUESDAY("No"),  
    WEDNESDAY("no"),  
    THURSDAY("yes"),  
    FRIDAY("YES!"),  
    SATURDAY("YEAH~!"),  
    SUNDAY("SAD...");  
  
    public String text; // field for custom message  
  
    Weekday(String text){  
        System.out.println(this + " was called!");  
        this.text = text; // assigning custom messages  
    }  
}
```

# Enumeration: Class-usage (cont'd)

## ■ Method

- Getter for a custom value
  - Encapsulation purpose

```
public enum Weekday {  
    MONDAY("NO"),  
    TUESDAY("No"),  
    WEDNESDAY("no"),  
    THURSDAY("yes"),  
    FRIDAY("YES!"),  
    SATURDAY("YEAH~!"),  
    SUNDAY("SAD...");  
  
    private String text;  
    Weekday(String text){  
        System.out.println(this + " was called!");  
        this.text = text;  
    }  
  
    public String getText() {  
        return text;  
    }  
}
```

```
switch (myDay) {  
    case MONDAY:  
    case FRIDAY:  
    case SATURDAY:  
    case SUNDAY:  
        System.out.println(myDay.name()+ " is "+ myDay.getText());  
        break;  
    default:  
        System.out.println("Midweek days are so-so.");  
        break;  
}
```

# Enumeration: Class-usage (cont'd)

---

## ■ Method

### ➤ Enum class methods

- `name()`: returns the defined name of an enum constant in string form
- `values()`: returns an array of enum type containing all the enum constants
- `valueOf()`: takes a string and returns an enum constant having the same string name



# Enumeration: Class-usage

## ■ Example)

- Two custom values assigned
- Fields and getters added
- Constructor changed

```
for(Weekday w: Weekday.values()) {  
    System.out.print(w.getCode());  
    System.out.print(w.getText());  
    System.out.println(w.name());  
}  
  
Weekday someday = Weekday.valueOf("THURSDAY");  
System.out.println(someday.getText());
```

```
public enum Weekday {  
    MONDAY(0,"NO"),  
    TUESDAY(1,"No"),  
    WEDNESDAY(2,"no"),  
    THURSDAY(3,"yes"),  
    FRIDAY(4,"YES!"),  
    SATURDAY(5,"YEAH~!"),  
    SUNDAY(6,"SAD...");  
  
    private int code;  
    private String text;  
  
    Weekday(int code, String text){  
        System.out.println(this + " was called!");  
        this.code = code;  
        this.text = text;  
    }  
  
    public String getText() {  
        return text;  
    }  
  
    public int getCode() {  
        return code;  
    }  
}
```



Exception Handling  
Enumeration

# Java Packages

# Java API Packages

---

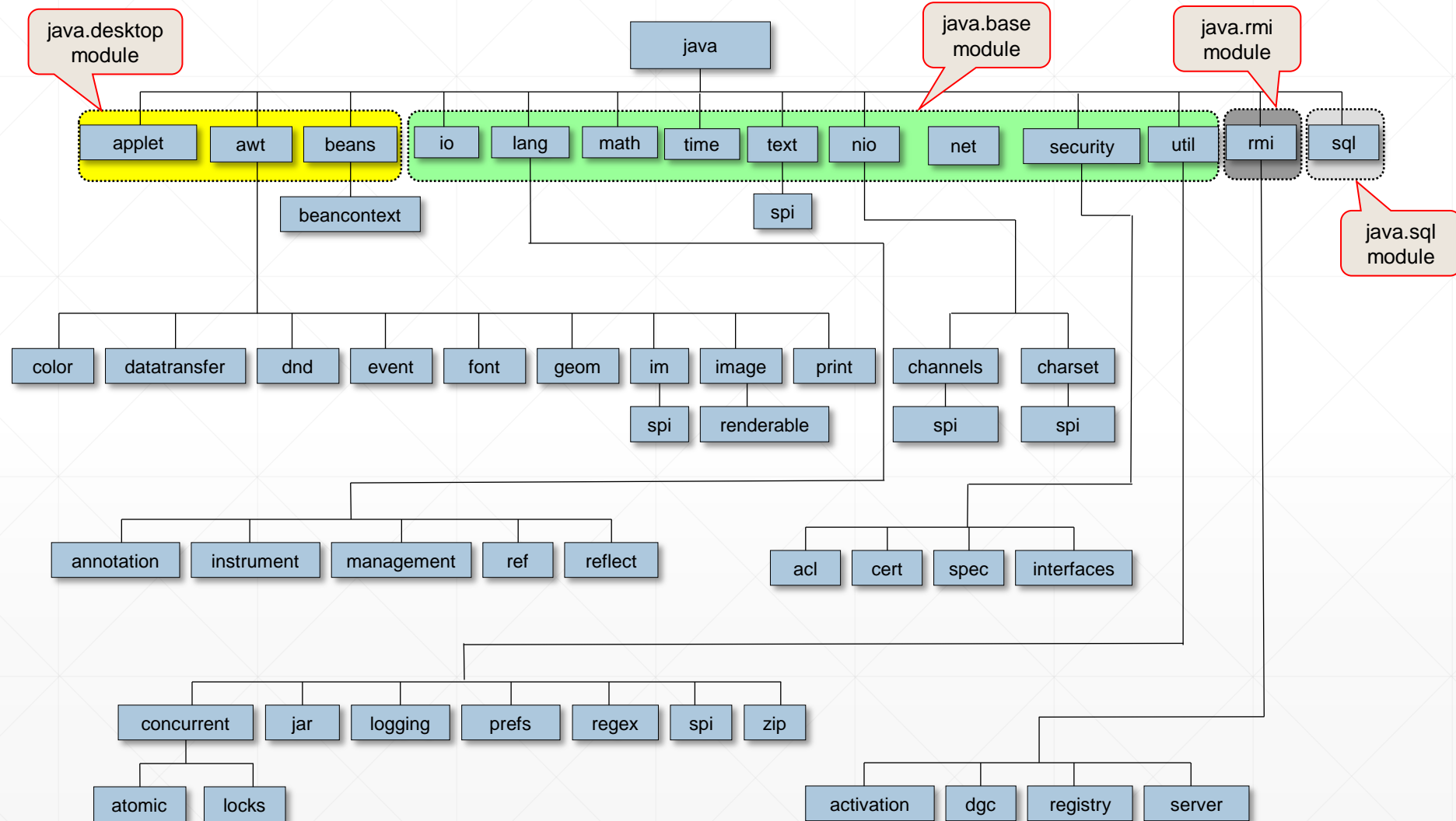
## ■ Java API (Application Programming Interface)

- Java's built-in software library to develop java programs
- A collection of frequently used classes and interfaces

## ■ Java API document (reference/specification)

- Document on how to use APIs
- Online reference
  - <https://docs.oracle.com/en/java/javase/11/docs/api/index.html>

# Java API Packages (cont'd)



# Java API Packages (cont'd)

---

## ■ java.lang

### ➤ Java language package

- Basic classes and interfaces for developing Java programs, including String, Math, etc.

## ■ java.util

### ➤ Utility package

- Various utility classes and interfaces including Date, Time, Vector, HashMap, etc.

## ■ java.io

### ➤ IO classes and interfaces for interacting with keyboard, monitor, printer, disk, etc.

## ■ java.awt

### ➤ Classes and Interfaces for Java GUI programming

## ■ javax.swing

### ➤ Swing package for Java GUI programming

## ■ ...

# Object class

## ■ Root class of Java

- All classes implicitly, automatically inherit Object class
- All classes can use the methods of Object class

## ■ Methods

Method	Description
<code>boolean equals(Object obj)</code>	Returns true if this object is the same as obj
<code>Class getClass()</code>	Returns the runtime class of this object
<code>int hashCode()</code>	Returns a hashcode value for this object
<code>String toString()</code>	Returns a string representation of this object

- ... and more!

# Object class (cont'd)

- Example) Get the class name, hashCode, string representation of Object

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
public class ObjectPropertyEx {  
    public static void print(Object obj) {  
        System.out.println(obj.getClass().getName()); // class name  
        System.out.println(obj.hashCode()); // hashCode  
        System.out.println(obj.toString()); // string representation  
        System.out.println(obj); // object itself  
    }  
    public static void main(String [] args) {  
        Point p = new Point(2,3);  
        print(p);  
    }  
}
```

# Object class (cont'd)

## ■ toString() method

- Returns a string representation of an object
- toString() method implementation of Object class

```
public String toString() {  
    return getClass().getName() + "@" + Integer.toHexString(hashCode());  
}
```

- Automatically invoked when a string manipulation with an object reference is required
- Overriding toString() for each class
  - Can return a class-specific string representation



# Object class (cont'd)

## ■ toString() method

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public String toString() {  
        return "Point(" + x + "," + y + ")";  
    }  
}  
  
public class ToStringEx {  
    public static void main(String [] args) {  
        Point p = new Point(2,3);  
        System.out.println(p.toString());  
        System.out.println(p);  
        System.out.println("Info: " + p);  
    }  
}
```

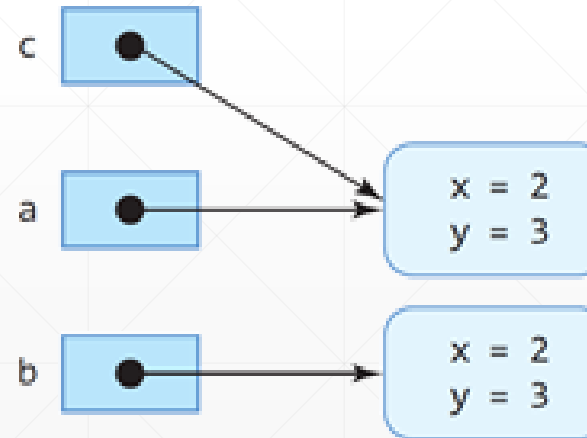
# Object class (cont'd)

## ■ equals() method

- Returns true if this object is the same as obj
- Object's equals() method basically works like == operator
  - == operator: returns true if two operands point the same address (for reference type)

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
}
```

```
Point a = new Point(2,3);  
Point b = new Point(2,3);  
Point c = a;  
if(a == b) // false  
    System.out.println("a==b");  
if(a == c) // true  
    System.out.println("a==c");
```



# Object class (cont'd)

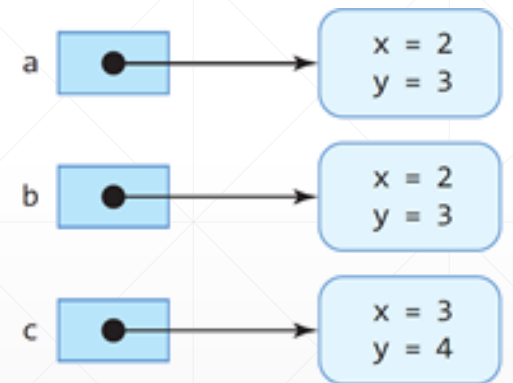
## ■ equals() method

- Can be overridden in the class to return true if this object is the same as obj in terms of contents, rather than address

```
class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public boolean equals(Object obj) {  
        Point p = (Point)obj;  
        if(x == p.x && y == p.y)  
            return true;  
        else return false;  
    }  
}
```

```
Point a = new Point(2,3);  
Point b = new Point(2,3);  
Point c = new Point(3,4);
```

```
if(a == b) // false  
    System.out.println("a==b");  
if(a.equals(b)) // true  
    System.out.println("a is equal to b");  
if(a.equals(c)) // false  
    System.out.println("a is equal to c");
```



# Object class (cont'd)

## ■ equals() method: Example

- Implement Rect class with width and height fields. If the areas of two Rect objects are same, then equals() method of Rect should return true.

```
class Rect {  
    private int width;  
    private int height;  
    public Rect(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    public boolean equals(Object obj) {  
        Rect p = (Rect)obj;  
        if (width*height == p.width*p.height)  
            return true;  
        else  
            return false;  
    }  
}
```

```
public class EqualsEx {  
    public static void main(String[] args) {  
        Rect a = new Rect(2,3);  
        Rect b = new Rect(3,2);  
        Rect c = new Rect(3,4);  
        if(a.equals(b))  
            System.out.println("a is equal to b");  
        if(a.equals(c))  
            System.out.println("a is equal to c");  
        if(b.equals(c))  
            System.out.println("b is equal to c");  
    }  
}
```

# Wrapper Class

- Dedicated class for primitive datatypes

- “Wrap” the primitive data type into an object of that class

Primitive	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
int	Integer
float	Float
double	Double
long	Long
short	Short

# Wrapper Class (cont'd)

## ■ Methods

➤ All wrapper classes have similar methods

## ■ Main methods of Integer class

Modifier and Type	Method	Description
boolean	<b>equals</b> ( <b>Object</b> obj)	Compares this object to the specified object
byte	<b>byteValue</b> ()	Returns the value of this Integer as a byte after a narrowing primitive conversion
double	<b>doubleValue</b> ()	Returns the value of this Integer as a double after a widening primitive conversion
float	<b>floatValue</b> ()	Returns the value of this Integer as a float after a widening primitive conversion
int	<b>intValue</b> ()	Returns the value of this Integer as an int
long	<b>longValue</b> ()	Returns the value of this Integer as a long after a widening primitive conversion
short	<b>shortValue</b> ()	Returns the value of this Integer as a short after a narrowing primitive conversion
static int	<b>sum</b> (int a, int b)	Adds two integers together as per the + operator
static int	<b>max</b> (int a, int b)	Returns the greater of two int values as if by calling <b>Math.max</b> .
static int	<b>min</b> (int a, int b)	Returns the smaller of two int values as if by calling <b>Math.min</b> .

# Wrapper Class (cont'd)

## ■ Main methods of Integer class (cont'd)

Modifier and Type	Method	Description
static int	<b>parseInt(String s)</b>	Parses the string argument as a signed decimal integer
static int	<b>parseInt(String s, int radix)</b>	Parses the string argument as a signed integer in the radix specified by the second argument
static <b>String</b>	<b>toBinaryString(int i)</b>	Returns a string representation of the integer argument as an unsigned integer in base 2
static <b>String</b>	<b>toHexString(int i)</b>	Returns a string representation of the integer argument as an unsigned integer in base 16
static <b>String</b>	<b>toOctalString(int i)</b>	Returns a string representation of the integer argument as an unsigned integer in base 8
<b>String</b>	<b>toString()</b>	Returns a String object representing this Integer's value
static <b>String</b>	<b>toString(int i)</b>	Returns a String object representing the specified integer
static <b>String</b>	<b>toString(int i, int radix)</b>	Returns a string representation of the first argument in the radix specified by the second argument
static <b>Integer</b>	<b>valueOf(int i)</b>	Returns an Integer instance representing the specified int value
static <b>Integer</b>	<b>valueOf(String s)</b>	Returns an Integer object holding the value of the specified String
static <b>Integer</b>	<b>valueOf(String s, int radix)</b>	Returns an Integer object holding the value extracted from the specified String when parsed with the radix given by the second argument

# Wrapper Class (cont'd)

## ■ Instantiation of wrapper class

---

static **Integer** **valueOf**(int i)

Returns an Integer instance representing the specified int value

static **Integer** **valueOf**(String s)

Returns an Integer object holding the value of the specified String

static **Integer** **valueOf**(String s, int radix)

Returns an Integer object holding the value extracted from the specified String when parsed with the radix given by the second argument

---

```
Integer i = Integer.valueOf(10);  
Character c = Character.valueOf('c');  
Double f = Double.valueOf(3.14);  
Boolean b = Boolean.valueOf(true);
```

```
Integer l = Integer.valueOf("10");  
Double d = Double.valueOf("3.14");  
Boolean b = Boolean.valueOf("false");
```

---

int **intValue()**

Returns the value of this Integer as an int

---

```
Integer i = Integer.valueOf(10);  
int ii = i.intValue(); // ii = 10  
  
Character c = Character.valueOf('c');  
char cc = c.charValue(); // cc = 'c'
```

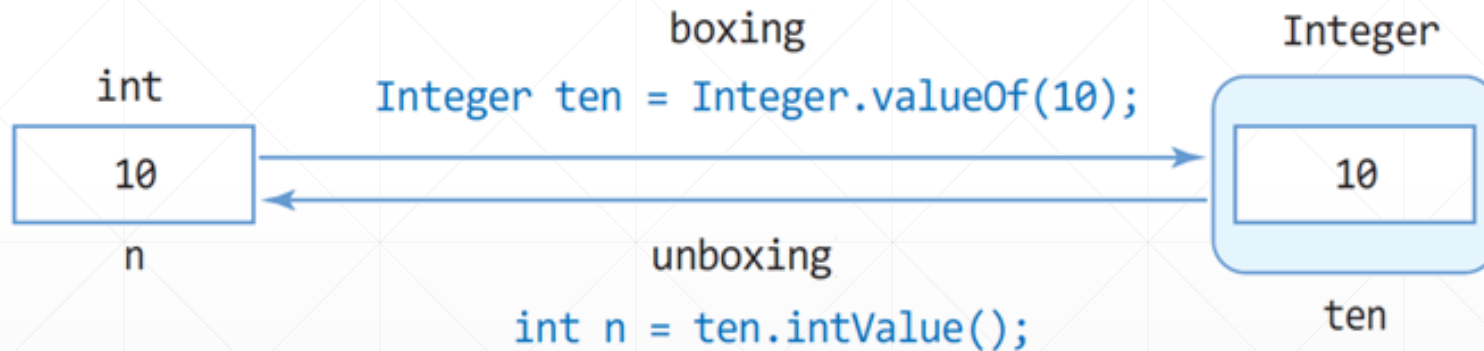
```
Double f = Double.valueOf(3.14);  
double dd = d.doubleValue(); // dd = 3.14  
  
Boolean b = Boolean.valueOf(true);  
boolean bb = b.booleanValue(); // bb = true
```



# Wrapper Class (cont'd)

## ■ Boxing & Unboxing

- Boxing: conversion from primitive types to wrapper classes
- Unboxing: conversion from wrapper classes to primitive types



- Automatic boxing & unboxing

```
Integer i = 10; // auto boxing (Integer.valueOf(10))
int ival = i; // auto unboxing (i.intValue())
System.out.println(ival);
```

# Wrapper Class (cont'd)

## ■ String $\leftrightarrow$ primitive types

static int	<b>parseInt(String s)</b>	Parses the string argument as a signed decimal integer
static int	<b>parseInt(String s, int radix)</b>	Parses the string argument as a signed integer in the radix specified by the second argument

```
int i = Integer.parseInt("123");           // i = 123
boolean b = Boolean.parseBoolean("true");  // b = true
double f = Double.parseDouble("3.14" );    // d = 3.14
```

<b>String</b>	<b>toString()</b>	Returns a String object representing this Integer's value
static <b>String</b>	<b>toString(int i)</b>	Returns a String object representing the specified integer

```
String s1 = Integer.toString(123);          // from integer 123 to string "123"
String s2 = Integer.toHexString(123);       // from integer 123 to HexString "7b"
String s3 = Double.toString(3.14);          // from double 3.14 to string "3.14"
String s4 = Character.toString('a');        // from character 'a' to string "a"
String s5 = Boolean.toString(true);         // from boolean true to string "true"
```

# Wrapper Class (cont'd)

---

- When to use?
  - Utility class for primitive types
  - Generic syntax
  - ...

# Q&A

---

## ■ Next week

➤ Generic & Collection