

CPE207 Object Oriented Programming

Week 4

Working with classes: Exception handling, static keyword



Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

These Slides mainly adopted from Assist. Prof. Dr. Ozacar Kasim lecture notes

The class notes are a compilation and edition from many sources. The instructor does not claim intellectual property or ownership of the lecture notes.

Encapsulation(Data Hiding): set() and get()

- ▶ Why do we create methods to access variables, if we can access them directly?
 - **Answer:** security issue.
- ▶ 'private' variables can only be accessed in the class. It's as if they were invisible out of the scope of the class / object.
- ▶ Using set() and get() methods, A class can have total control over what is stored in its attributes

```
Class Foo {  
    private String name;  
  
    public void setName(String name){  
        //put any condition  
        this.name = name;  
    }  
  
    public String getName(){  
        //put any condition  
        return name;  
    }  
}
```

Benefits of Encapsulation

- ▶ A class can have total control over what is stored in its attributes.
- ▶ Attributes of a class can be made read-only or write-only.
- ▶ The users of a class do not know how the class stores its data.
 - A class can change the data type of a attributes and users of the class do not need to change any of their code.

errors

Compile Time Errors

- Syntax error in a;
- Type mismatch, wrong casting
 - int x = "asdsad";
 - ...

Run Time Errors

Exceptions

- exceptions are events that disrupt the normal flow of the program's instructions **during the execution of a program.**

We must handle the Exception, otherwise the program will crash!!

Exception Example

```
System.out.println("Enter a value");  
int value1=scanner.nextInt();
```

```
System.out.println("Enter another value");  
int value2=scanner.nextInt();
```

What if value2 is zero? This is an exception

```
System.out.println(value1/value2);
```

Value1 /value2 is undefined

This is an exception that disrupts the
normal flow of the program's
instructions

Exception Example

```
System.out.println("Enter a value");
```

```
int value1=scanner.nextInt();
```

```
System.out.println("Enter another value");
```

```
int value2=scanner.nextInt();
```

```
System.out.println(value1/value2) ;
```

try

{

}

```
catch (Exception e)
```

f

```
System.out.println("you cannot divide by zero");
```

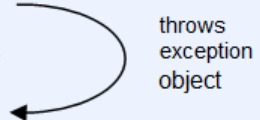
}

Exception Handling : Basic Concepts

- ▶ Exception handling is a mechanism that allows you to take appropriate action **to avoid run-time errors**.
- ▶ Java provides five keywords to support exception handling.
 - **Try** : The try block contain statements which may generate exceptions.
 - **Catch** :The catch block defines the action to be taken, when an exception occur.
 - **Throw** : When an exception occur in try block, it is thrown to the catch block using throw keyword.
 - *Throws* : *Throws keyword is used in situation, when we need a method to throw an exception.*
 - **Finally** : If exception occur or not, finally block will always execute.

The general form of try-catch block in Java.

```
try
{
    -----
    Runtime error occur
    -----
}
catch (Exception Ex )
{
    -----
    -----
}
```



throws exception object

Finally block can be used to put "cleanup" code such as closing a file, closing connection etc.

Exception Handling: Syntax

```
try{  
    //statements that may cause an exception  
}  
catch (Exception(type) e(object)){  
    //error handling code  
    //System.out.println(e.getMessage());  
}
```

```
public String getMessage()
```

–Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.

Example for Exception Handling : Time1 class

- Class Time1 represents the time of day.
- **private int** instance variables **hour, minute and second** represent the time in universal-time format (24-hour clock format in which hours are in the range 0-23, and minutes and seconds are each in the range 0-59).
- public methods **setHour(), setMinute(), setSecond(), toUniversalString()** and **toString()**.

Time1
-hour:int -minute:int -second:int
<<constructor>>Time1(hour:int, minute: int, second:int) +setHour(h: int) +setMinute(m: int) +setSecond(s: int) +toString():String +toUniversalString():Sting

Time1 class : AM/PM vs 24-H

```
public class Time1 {  
    private int hour;    // 0 - 23  
    private int minute; // 0 - 59  
    private int second; // 0 - 59  
  
    public Time1(int hour, int minute, int second) {  
        setHour(hour);  
        setMinute(minute);  
        setSecond(second);  
    }  
  
    public void setHour( int h ){  
        if (h >= 0 && h < 24)  
            this.hour =h;  
    }  
  
    public void setMinute( int m ){  
        if (m >= 0 && m < 60)  
            this.minute =m;  
    }  
}
```

```
    public void setSecond(int s ){  
        if (s >= 0 && s < 60)  
            this.second =s;  
    }  
    // convert to String in universal-time format (HH:MM:SS)  
    public String toUniversalString()  
    {  
        return String.format( "%02d:%02d:%02d", hour, minute, second );  
    } // end method toUniversalString  
  
    public String toString()  
    {  
        return String.format( "%d:%02d:%02d %s",  
            ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),  
            minute, second, ( hour < 12 ? "AM" : "PM" ) );  
    } // end method toString  
    // end class Time1
```

Time1 class : Exception Handling

```
public class Time1 {
    private int hour; //0-23
    private int minutes; // 0-59
    private int seconds; // 0-59

    Time1(int hour, int minutes, int seconds){
        setHour(hour);
        setMinutes(minutes);
        setSeconds(seconds);
    }

    void setHour(int hour){
        if(hour<0 || hour>23)
            throw new IllegalArgumentException("Wrong Hour");
        this.hour = hour;
    }

    void setMinutes(int minutes){
        if(minutes<0 || minutes>59)
            throw new IllegalArgumentException("Wrong minutes");
        this.minutes = minutes;
    }

    void setSeconds(int seconds){
        if(seconds<0 || seconds>59)
            throw new IllegalArgumentException("Wrong seconds");
        this.seconds = seconds;
    }

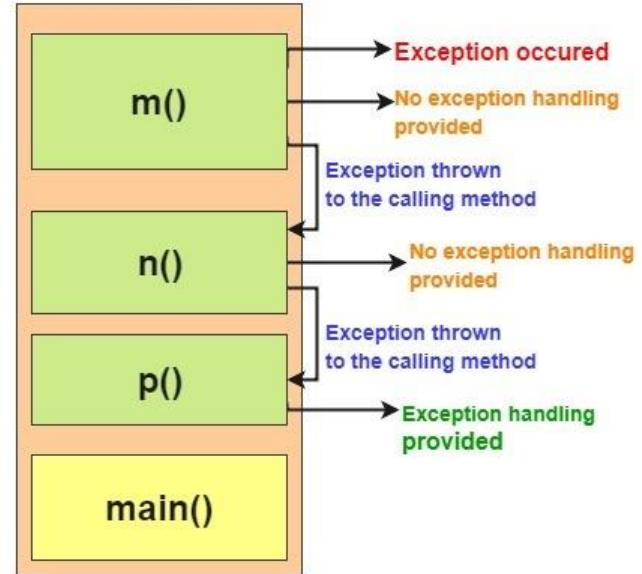
    public String printTime(){
        return String.format("%02d:%02d:%02d",hour,minutes,seconds);
    }
}
```

```
public class Week4gr2exp2 {
    public static void main(String[] args) {
        try{
            Time1 t1 = new Time1(24,67,56);
            System.out.println(t1.printTime());
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
        finally{
            System.out.println("Done");
        }
        System.out.println("The end of the prog");
    }
}
```

Java Exception propagation

```
class TestClass{  
    void m(){  
        int data=50/0;  
    }  
    void n(){  
        m();  
    }  
    void p(){  
        try{  
            n();  
        }catch(Exception e){  
            System.out.println("exception handled");  
        }  
    }  
    public static void main(String args[]){  
        TestClass obj=new TestClass();  
        obj.p();  
        System.out.println("normal flow...");  
    }  
}
```

Memory Stack



<https://simplenippets.tech/throw-throws-in-java-exception-handling-part-3/>

An example

If there is no error in input, then
no exception is thrown, and the
output will be

DONE

If there is an error in input, one
of the two exceptions is thrown,
and the output will be

Not an integer

DONE

or

Error: Out of bound

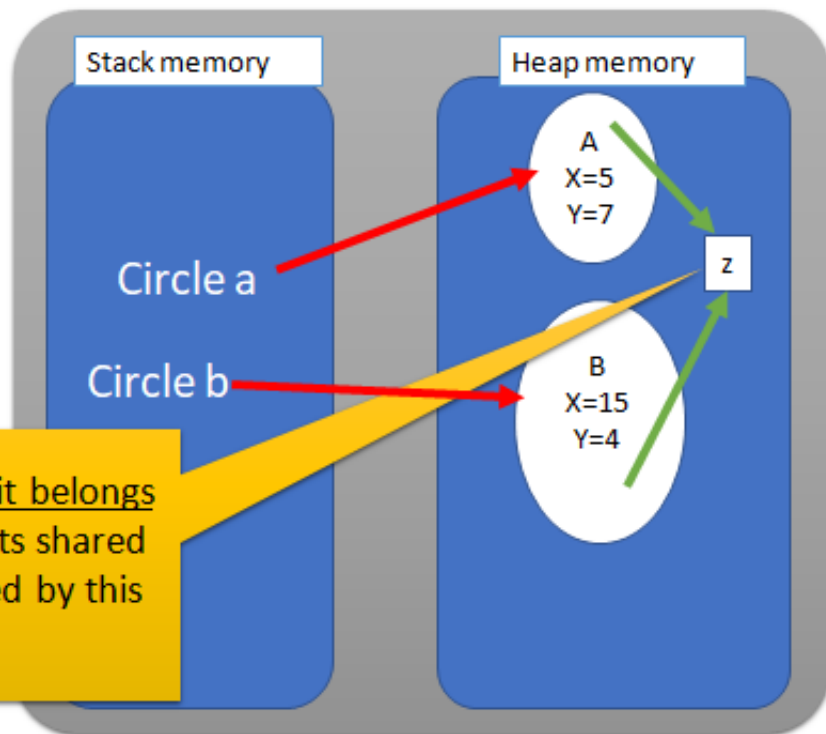
DONE

```
Scanner scanner = new Scanner(System.in);
    try{
        int num = scanner.nextInt();
        if (num > 100) {
            throw new Exception("Out of bound");
        }
    } catch (InputMismatchException e) {
        System.out.println("Not an
integer");
    } catch (Exception e) {
        System.out.println("Error: "+
e.getMessage());
    } finally {
        System.out.println("DONE");
    }
}
```

Static keyword

- ▶ The **static keyword** is a **non-access modifiers** and used in java mainly for **memory management**. It is used with variables, methods, blocks and nested classes.
- ▶ It is a keyword that **are used for share the same variable or method of a given class.**
- ▶ This is used for a constant variable or a method that is the same for every instance of a class.

```
class Circle{  
    private int x;  
    private int y;  
    static int z;  
}
```



Here `z` is static variable: it belongs to class, not an object. Its shared by all the objects created by this class.

Static variable

- ▶ If any variable, we declared as static is known as static variable.
- ▶ Static variable is used for fulfill the common requirement. **For Example**, college name of students etc.
Name of the college is common for all students.
- ▶ The static variable **allocate memory only once** in class area at the time of class loading.
- ▶ Using static variable we make our program memory efficient (i.e it saves memory).

When and why we use static variable?

- ▶ Suppose we want to store record of all employee of any company, in this case employee id is unique for every employee but company name is common for all. So, use static variable to store the company name.
- ▶ When we create a static variable as a company name then only once memory is allocated otherwise it allocate a memory space each time for every employee object.

An example of static keyword

```
public class Employee {  
    public static String companyName = "MNG";  
    private int id;  
    private String name;  
    static int number;  
  
    public Employee(int id, String name){  
        this.id = id;  
        this.name = name;  
        number++;  
    }  
  
    public void getInfo(){  
        System.out.println(this.id + " " + this.name + " " + Employee.companyName );  
    }  
}
```

final Keyword

- ▶ static keyword always fixed the memory that means that *will be located only once in the program*
- ▶ final keyword always **fixes the value** that means it makes variable values constant
- ▶ Note: As for as real time statement there concern every final variable should be declared the static but there is no obligation that every static variable declared as final.

Thanks 😊

Lab Exercise 1: Exception Handling

- ▶ You need to read an integer value from keyboard. However, let's say a user typed a string value.
- ▶ This will cause an input mismatch exception.
- ▶ Handle this exception.
(Hint: use `InputMismatchException` exception.)

Lab Exercise 2: Creating your own Exception

- Create a Worker class which contains name salary attributes
- Create setName and setSalary methods.
- Your program should throw an exception when salary value is less than zero. (When you enter an illegal argument)
 - The exception message is “salary amount must be greater than zero”
- In main method handle the exception and display the exception message.

- ▶ Add another attribute (counter) which must be shared by all the instances of Worker class.
- ▶ Create three workers and display number of total workers using counter variable.