ADVANCED OOP: EXCEPTIONS

OVERVIEW

OVERVIEW

- **Effective error handling is essential for building large software systems**
  - Want to detect when something goes wrong
  - Want to correct errors when possible
  - Want to abort the program when fatal errors occur

- **Traditional methods use if-statements to detect errors**
  - We can print error message and exit
  - We can loop asking user for correct input
  - We can have functions and methods return “status codes” to indicate if there was an error or not

OVERVIEW

- **Java provides language support for error handling**
  - Exception objects are used to describe exactly what kind of problem was detected
  - New “try throw catch” syntax is used to modify the normal program control when errors are detected

- **Lesson objectives:**
  - Show how exceptions are defined
  - Show how exceptions are “thrown”
  - Show how exceptions are “caught”
  - Show several example programs using exceptions

ADVANCED OOP: EXCEPTIONS

PART 1

THROWING EXCEPTIONS

ERROR HANDLING

- **Let’s revisit the Time class**
  - Private variables for hour, minute, second
  - Constructor functions
  - Get and set methods
  - Read and print methods

- **What should happen if someone enters an invalid time?**
  - We could print a message and abort
  - We could ask the user to try again
  - We could correct the invalid time
  - We could “throw an exception”

ERROR HANDLING

```
public class Time
{
    private int hour;
    private int minute;
    private int second;

    public Time()
    {
        hour = 0;
        minute = 0;
        second = 0;
    }

    public void setHour(int h) { hour = h; }
    public int getHour() { return hour; }
    ...
```

ERROR HANDLING

```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour in [0..23]: ");
   int hour = scnr.nextInt();

   // Check hour value
   if (hour < 0 || hour > 23)
   {
      System.out.println("Error: invalid hour");
      System.exit(-1);
   }
   ...
}
```

This leaves the program and gives error code of -1 to operating system

ERROR HANDLING

```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour in [0..23]: ");
   int hour = scnr.nextInt();

   // Check hour value
   while (hour < 0 || hour > 23)
   {
      System.out.print("Enter hour in [0..23]: ");
      hour = scnr.nextInt();
   }
   ...
}
```

This loop continues until user enters a valid hour

ERROR HANDLING

```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour in [0..23]: ");
   int hour = scnr.nextInt();

   // Check hour value
   if (hour < 0)
      hour = 0;
   if (hour > 23)
      hour = 23;
   ...
}
```

This sets the hour to the closest correct value

THROWING EXCEPTIONS

```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour in [0..23]: ");
   int hour = scnr.nextInt();

   // Check hour value
   if (hour < 0 || hour > 23)
        throw new Exception("Error detected");
   ...
}
```

This leaves the read method and returns the Exception object containing the error message to the calling function

THROWING EXCEPTIONS

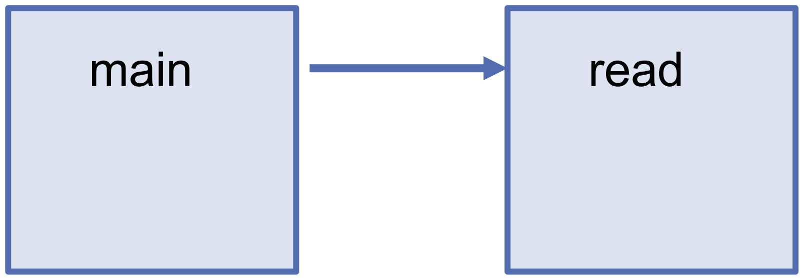
```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour in [0..23]: ");
   int hour = scnr.nextInt();

   // Check hour value
   if (hour < 0 || hour > 23)
        throw new IllegalArgumentException(
        "Hour not in [0..23] range");
   ...
}
```

This throws a more specific exception type and more detailed error message to the calling function

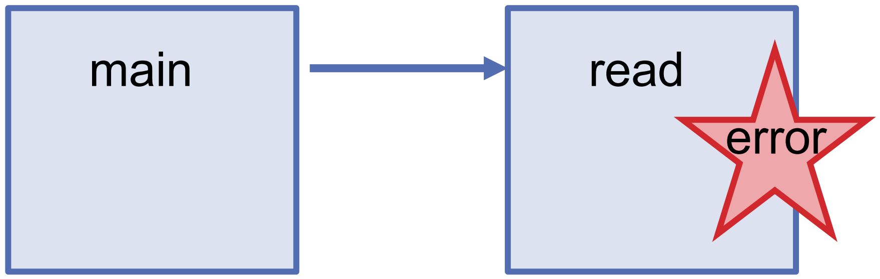
THROWING EXCEPTIONS

- **How does this work?**
  - The main program calls read



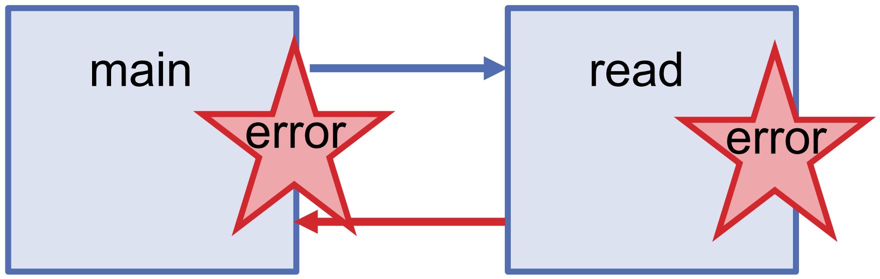
THROWING EXCEPTIONS

- **How does this work?**
  - The main program calls read
  - An error is detected



THROWING EXCEPTIONS

- **How does this work?**
  - The main program calls read
  - An error is detected
  - An exception is “thrown” to main



THROWING EXCEPTIONS

- **What does the main program do with the exception?**
  - Default: Throw the exception to operating system
  - This will end the program with an error message

```
Exception in thread "main"
java.lang.IllegalArgumentException: Hour not in [0..23] range
    at Time.read(Time.java:75)
    at Time.main(Time.java:128)
```

  - This is slightly better than printing an error message and exiting because we can see the "call stack"
  - We can do better than this …

ADVANCED OOP: EXCEPTIONS

PART 2

CATCHING EXCEPTIONS

CATCHING EXCEPTIONS

- **How can we detect and process the exception?**
  - Use the Java “try catch” syntax
  - Put code that could throw exceptions inside a “try block”
  - Use a “catch block” to process any exceptions that occur

```
try
{
   // run some code here
}
catch (Exception e)
{
   // handle any exceptions that occur
}
```

CATCHING EXCEPTIONS

```
public static void main(String[] args)
{
   Time time = new Time();
   try
   {
      time.read();
      time.print();
   }
   catch (IllegalArgumentException e)
   {
      String message = "Error: " + e.getMessage();
      System.out.println(message);
   }
   ...
```

$\leftarrow$ This try block has method calls that could cause an exception to be thrown

CATCHING EXCEPTIONS

```
public static void main(String[] args)
{
   Time time = new Time();
   try
   {
      time.read();
      time.print();
   }
   catch (IllegalArgumentException e)
   {
      String message = "Error: " + e.getMessage();
      System.out.println(message);
   }
   ...
```

If an exception does occur the program jumps immediately to the catch block and the exception is stored in e

CATCHING EXCEPTIONS

```
public static void main(String[] args)
{
   Time time = new Time();
   try
   {
      time.read();
      time.print();
   }
   catch (IllegalArgumentException e)
   {
      String message = "Error: " + e.getMessage();
      System.out.println(message);
   }
   ...
```

We can get the error message from the exception using the getMessage() method

CATCHING EXCEPTIONS

```
public static void main(String[] args)
{
   Time time = new Time();
   try
   {
      time.read();
      time.print();
   }
   catch (Exception e)
   {
      String message = "Error: " + e.getMessage();
      System.out.println(message);
   }
   ...
```

If several types of exceptions are possible, we can use a more generic exception type

CATCHING EXCEPTIONS

- **What happens when we have a sequence of method calls?**

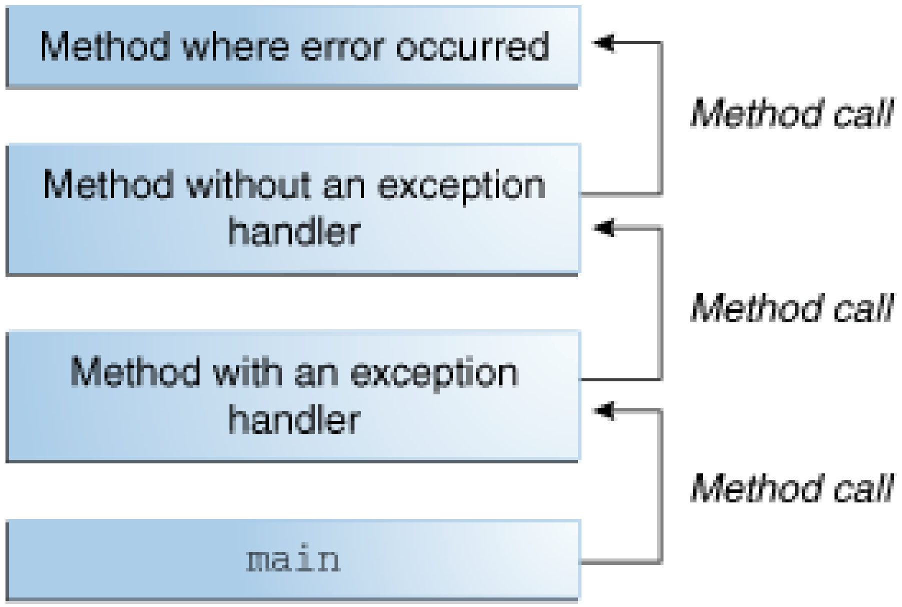

Method where error occurred
Method call
Method without an exception handler
Method call
Method with an exception handler
Method call
main

Figure from: docs.oracle.com

CATCHING EXCEPTIONS

- **The exception is thrown to the calling function**

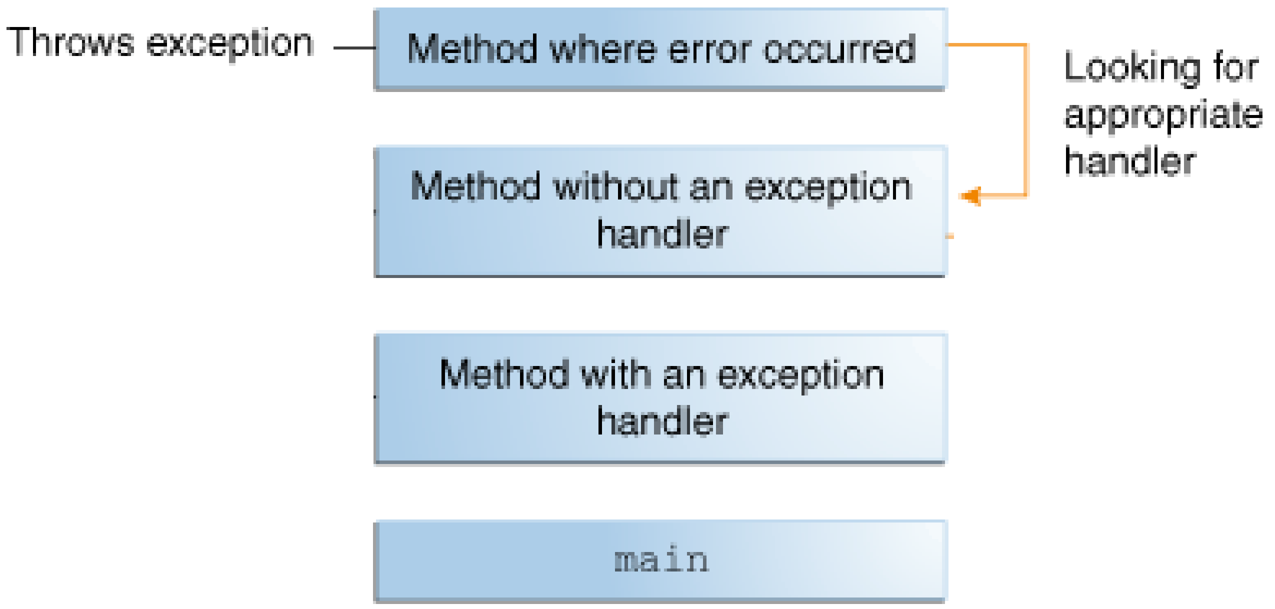

Throws exception
Method where error occurred
Method without an exception handler
Method with an exception handler
main
Looking for appropriate handler

Figure from: docs.oracle.com

CATCHING EXCEPTIONS

- There is no try-catch so the exception is thrown again

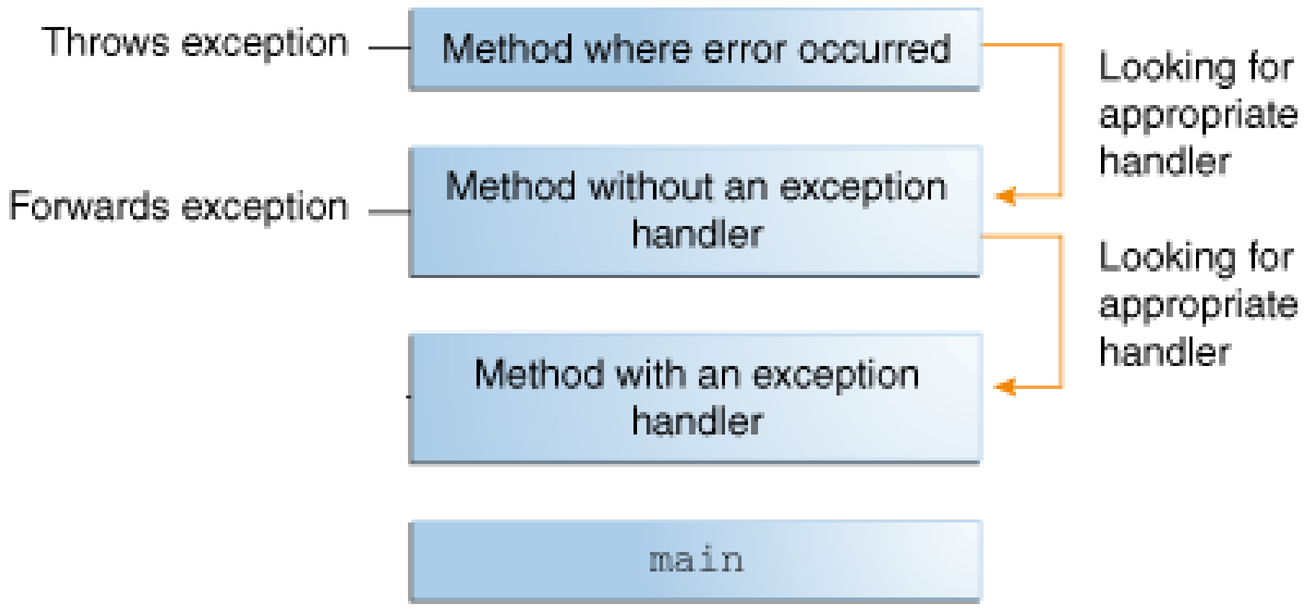
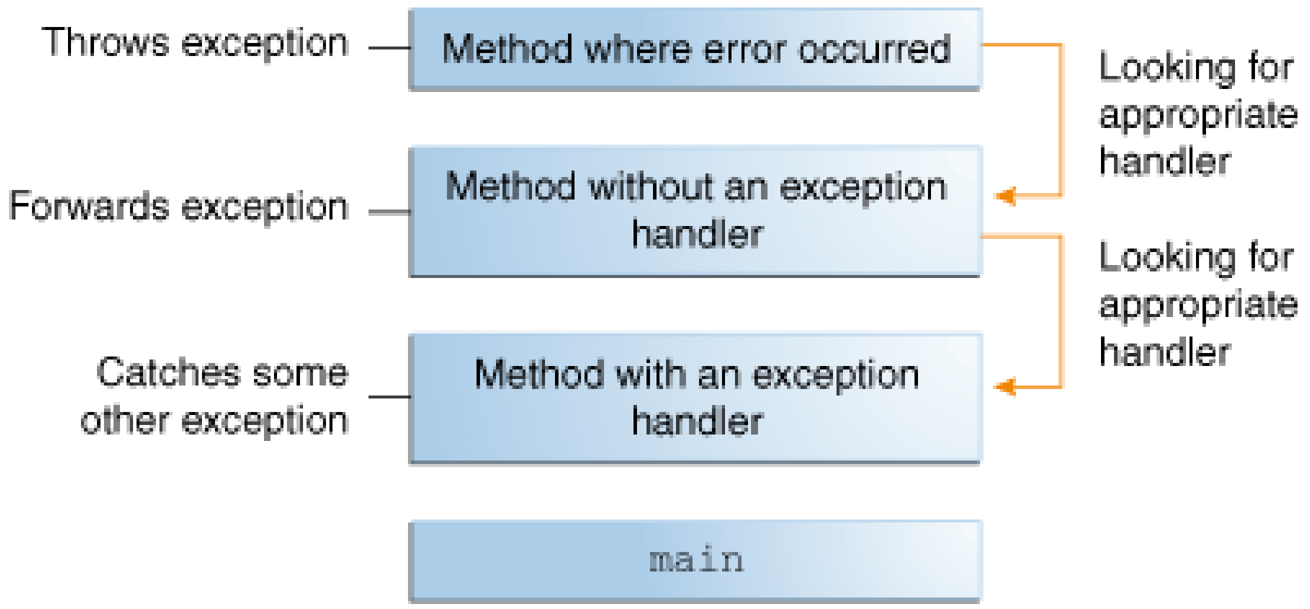

Throws exception
Method where error occurred
Forwards exception
Method without an exception handler
Method with an exception handler
main
Looking for appropriate handler
Looking for appropriate handler

Figure from: docs.oracle.com

CATCHING EXCEPTIONS

- **We can now handle exception and print message**



Throws exception
Method where error occurred
Forwards exception
Method without an exception handler
Catches some other exception
Method with an exception handler
main
Looking for appropriate handler
Looking for appropriate handler

Figure from: docs.oracle.com

CATCHING EXCEPTIONS

```
public static void main(String[] args)
{
   Time time = new Time();
   try
   {
      time.read();
      time.print();
   }
   catch (Exception e)
   {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
   }
   ...
```

We can print the stack trace ourselves using this method

CODE DEMO

Time1.java

Time2.java

CATCHING EXCEPTIONS

Output from Time1.java:

```
Testing the Time1 class
Enter hour: 11
Enter minute: 22
Enter second: 333
Exception in thread "main"
java.lang.IllegalArgumentException: Second not in [0..59]
range
        at Time1.setSecond(Time1.java:50)
        at Time1.read(Time1.java:79)
        at Time1.main(Time1.java:123)
```

CATCHING EXCEPTIONS

Output from Time2.java:

```
Testing the Time2 class
Enter hour: 11
Enter minute: 22
Enter second: 333
Error: Second not in [0..59] range
java.lang.IllegalArgumentException: Second not in [0..59]
range
        at Time2.setSecond(Time2.java:50)
        at Time2.read(Time2.java:79)
        at Time2.main(Time2.java:125)
```

SUMMARY

- **There are two categories of exceptions in Java**
- **Unchecked exceptions**
    - These exceptions that are not checked at compiled time, so the method throwing the exception does not need to handle or specify the exception. It is up to the programmers to specify or catch the exceptions.
- **Checked exceptions**
    - These exceptions are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

SUMMARY

- **Common unchecked exceptions**
  - IllegalArgumentException // see Time examples
  - ArrayIndexOutOfBoundsException
  - NullPointerException
  - NumberFormatException
  - AssertionError
  - StackOverflowError
- **See the Java documentation for the full list**
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

SUMMARY

- **Common checked exceptions**
  - IOException // see I/O examples
  - FileNotFoundException
  - ClassNotFoundException
  - InstantiationException
  - NoSuchMethodException
  - NoSuchFieldException
- **See the Java documentation for the full list**
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

SUMMARY

- **In this section described the Java syntax for exceptions**
  - How to detect errors and “throw” exceptions
  - How to call methods in a “try block”
  - How to handle exceptions in a “catch block”

- **Final Comments**
  - Exception handling was invented 50 years ago and is available in many programming languages (Java, C++, C#, Python)
  - When used properly exceptions can simplify error handling in many software applications
  - Unfortunately exceptions "create hidden control-flow paths that are difficult for programmers to reason about" (Weimer, 2008)