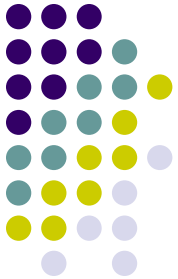
第五章 异常处理



5.1 错误处理的方法概述

5.2 Java的异常处理机制

5.3 创建自己的异常类

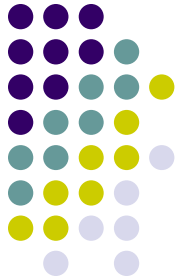
异常的演示

- 异常 （Exception）：发生于程序执行期间，表明出现了一个非法的运行状况。许多JDK中的方法在检测到非法情况时，会抛出一个异常对象。
- 例如：数组越界和被0除。

```
int i=1, j=0, k;
k=i/j;
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at AboutException.main(AboutException.java:9)
```

5.1 错误处理的方法概述



- 传统的程序运行时错误处理
  - 如C语言：
  - 函数返回值=某个可能会执行失败的函数（）；

```
if (函数返回值==表示该函数执行成功的值）{正常代码}
  else if (函数返回值==代表错误情况1的值) {处理错误情形1}
  else if (函数返回值==代表错误情况2的值) {处理错误情形2}
   ……
```

5.1 错误处理的方法概述

- 函数返回值=某个可能会执行失败的函数（）；
-

if(函数返回值!=表示该函数执行成功的值）
{ Switch(函数返回值){
case 错误情况1的值:处理错误情形1
case 错误情况2的值:处理错误情形2
……}}
else{正常代码}

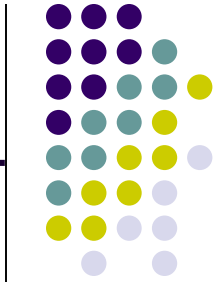
5.1 错误处理的方法概述

- 缺点：
  - 整个程序代码穿插错误处理代码，使得条理性和可读性差；
  - 对错误处理程序难以集中管理，难以保证程序的一致性；
  - 对于返回值的意义，要借助于文档，程序维护困难。

5.1 错误处理的方法概述

- 异常处理
  - 在进行程序设计时，错误的产生是不可避免的。
  - 所谓错误，是在程序运行过程中发生的异常事件，这些事件的发生将阻止程序的正常运行
  - 如何处理错误？把错误交给谁去处理？程序又该如何从错误中恢复？
  - 为了加强程序的鲁棒性，Java语言具有特定的运行错误处理机制。

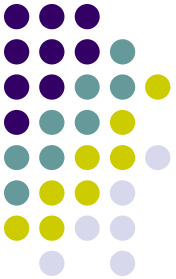
5.1 错误处理的方法概述



如C++，Java语言：

- 在异常发生时，由编程语言提供的某种机制通知应用程序，让应用程序决定如何进行下一步的处理。
- 传统方式：
  - 负责测出错误的发生 （程序设计者）
  - 进行错误的处理
- 异常处理方式：
  - 进行错误的处理 （程序设计者）

第五章 异常处理

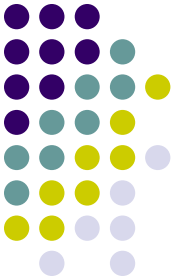


5.1 错误处理的方法概述

5.2 Java的异常处理机制

5.3 创建自己的异常类

5.2 Java的异常处理机制



- 错误有三类：语法错误、运行错误和逻辑错误
  - 出现语法错误（syntax error）的原因是没有遵循语言的规则，它们可以由编译器检查发现。
  - 在程序运行过程中，如果环境发现了一个不可能执行的操作，就会出现运行错误（runtime error）。
  - 如果程序没有按照预期的方案执行，就会发生逻辑错误（logic error）。

5.2 Java的异常处理机制

运行错误

如果这里出错（如用户输入不是一个整数），则会发生异常。

发生异常后，就会跳过后面的内容，并终止程序

```
import java.util.Scanner;
public class RuntimeExceptionDemo
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter a integer:");

        // 从键盘读取用户输入，并转换为一个整数，
        // 赋值给 number 这个变量
        int number = scanner.nextInt();

        System.out.println("Your input is: " + number);
    }
}
```

5.2 Java的异常处理机制

- 捕获运行错误
- 运行错误不是我们想要的，它会引起程序异常终止。
- 需要有某种手段来捕获这个错误，让程序在收到错误后，能够继续执行
- 对于上一个例子来说，解决方案：

如果发现用户输入了错误的内容（如输入的不是一个整数），则提醒用户再次输入，直到正确为止

5.2 Java的异常处理机制--改进版

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean inputIsValid = false;

        while (!inputIsValid)
        {
            try
            {
                System.out.println("Please enter a integer:");
                int number = scanner.nextInt();
                System.out.println("Your input is: " + number);
                inputIsValid = true;
            }
            catch (InputMismatchException ex)
            {
                System.out.println("Incorect input, please try again.");
                scanner.nextLine(); // 重新开始接受输入
            }
        }
    }
}
```

如果在该行出现异常，在try块中的其他部分被跳过，并转到catch块

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine(); // 重新开始接受输入
         }
      }
   }
}
```

开始接收用户输入。
假设用户输入的不是整数，如 “abc”

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine(); // 重新开始接受输入
         }
      }
   }
}
```

出错点

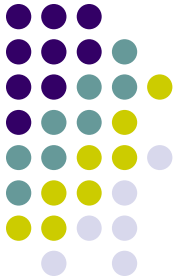
try块中剩余代码没有被执行

发生异常，并被catch捕获

跟踪程序执行

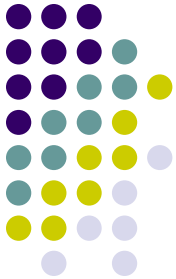
```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine(); // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行



```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行



```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

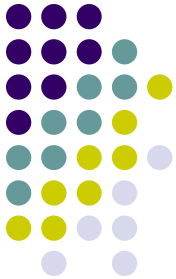
```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine(); // 重新开始接受输入
         }
      }
   }
}
```

接收用户输入。
假设用户输入的是整数，如 100

跟踪程序执行

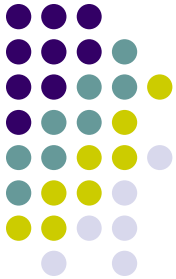
```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行



```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行



```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

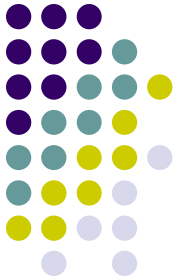
跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

跟踪程序执行

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class RuntimeExceptionDemo2
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);
      boolean inputIsValid = false;

      while (!inputIsValid)
      {
         try
         {
            System.out.println("Please enter a integer:");
            int number = scanner.nextInt();
            System.out.println("Your input is: " + number);
            inputIsValid = true;
         }
         catch (InputMismatchException ex)
         {
            System.out.println("Incorect input, please try again.");
            scanner.nextLine();  // 重新开始接受输入
         }
      }
   }
}
```

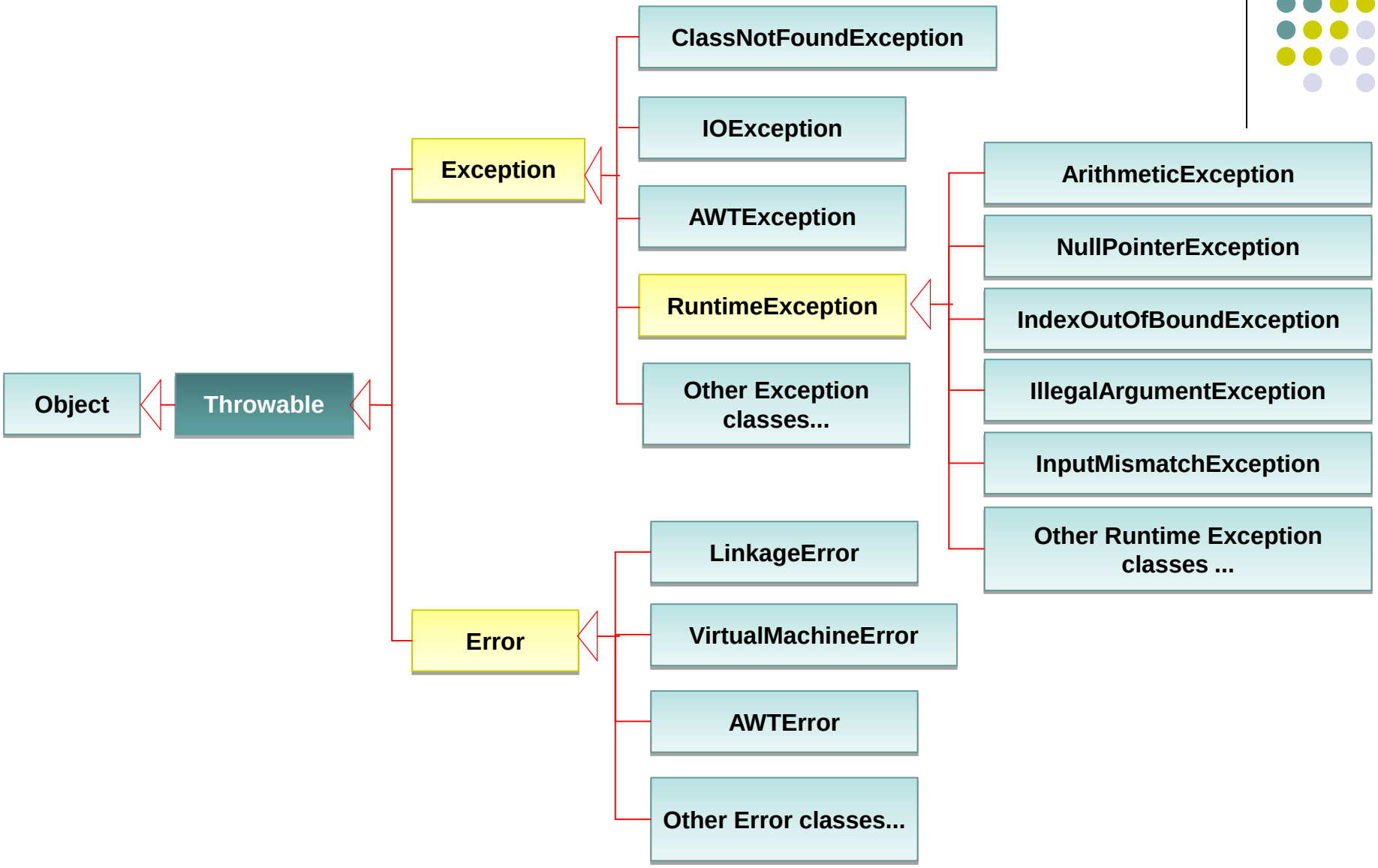
5.2 Java的异常处理机制



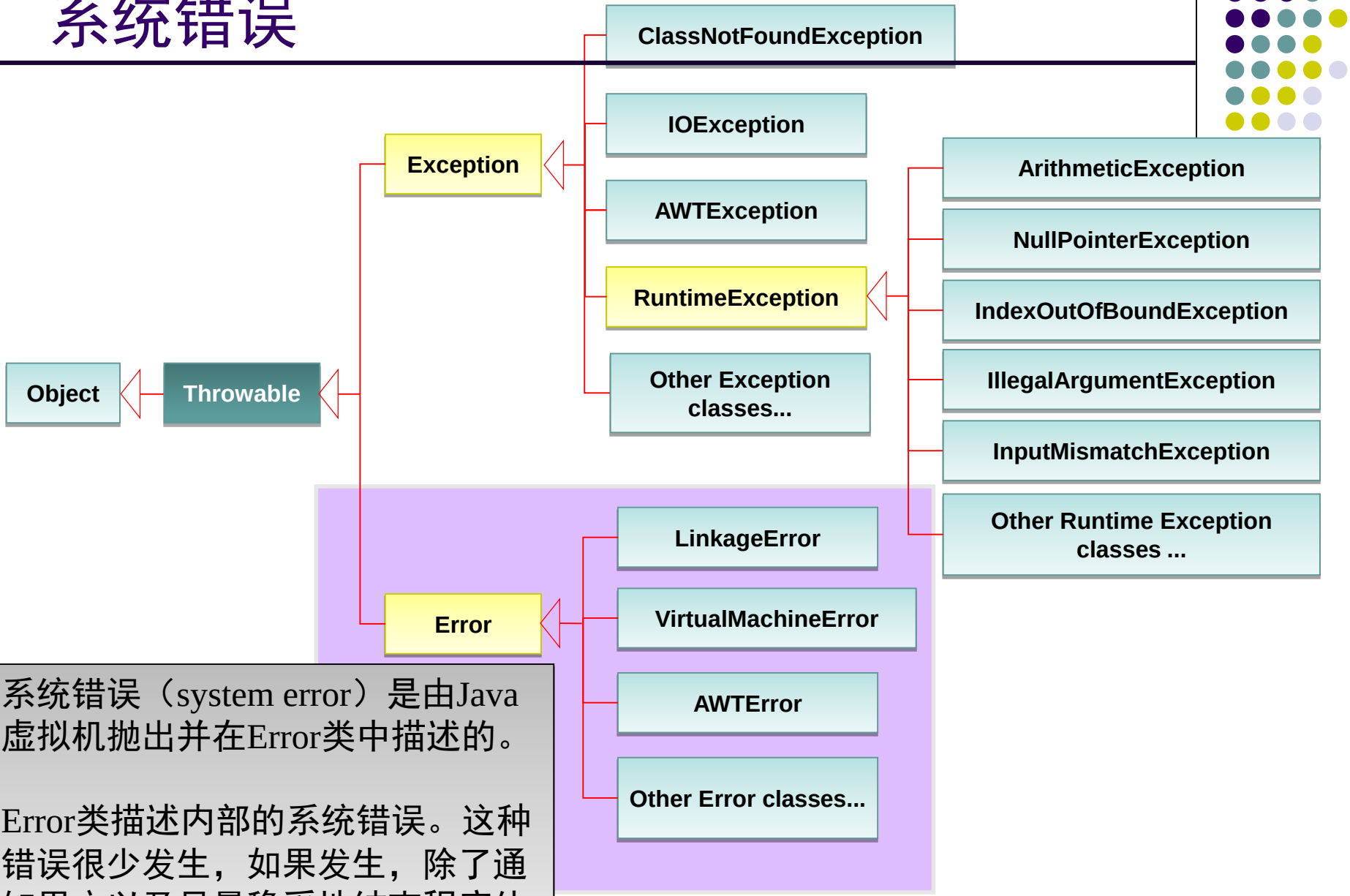
- 异常处理的目的是依据实际情况提供不同的错误应对策略与手段，使程序更稳定，更安全。
- 异常处理的主要用途是提供准确的错误消息，解释失败的原因、位置和错误类型等，同时提供一定的恢复能力，尽可能地保证数据完整性不被破坏，并让程序能继续运行。
- Java使用try/catch结构来捕获异常

```
try
{
        程序正常逻辑
}
catch (...)
{
        异常处理代码
}
```

异常类继承结构图


Object
Throwable
Exception
Error
ClassNotFoundException
IOException
AWTException
RuntimeException
Other Exception classes...
ArithmeticException
NullPointerException
IndexOutOfBoundException
IllegalArgumentException
InputMismatchException
Other Runtime Exception classes ...
LinkageError
VirtualMachineError
AWTError
Other Error classes...

系统错误

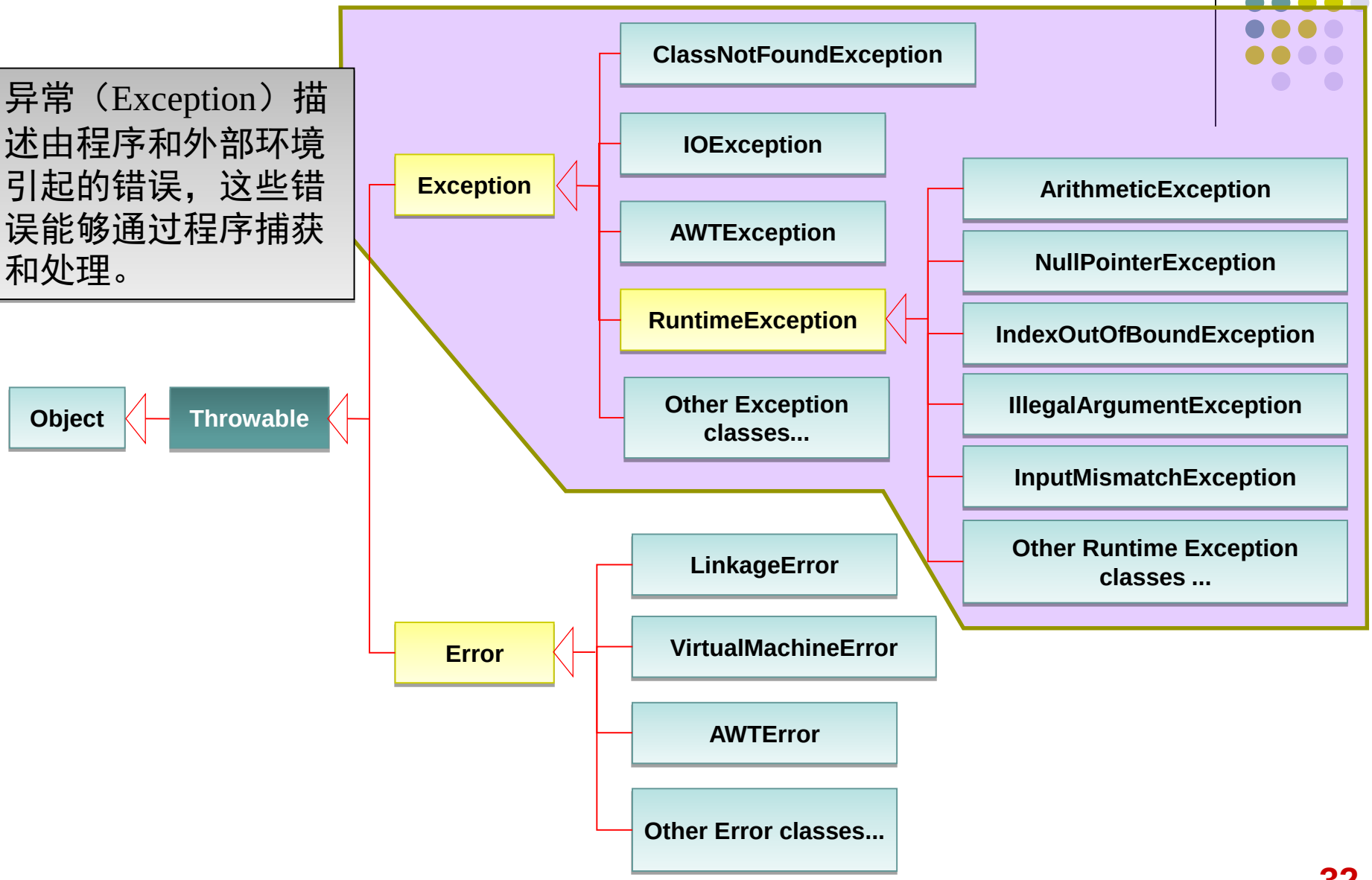


系统错误（system error）是由Java虚拟机抛出并在Error类中描述的。

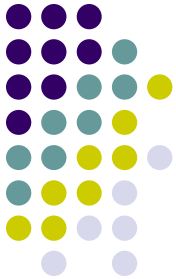
Error类描述内部的系统错误。这种错误很少发生，如果发生，除了通知用户以及尽量稳妥地结束程序外，几乎什么也不能做。

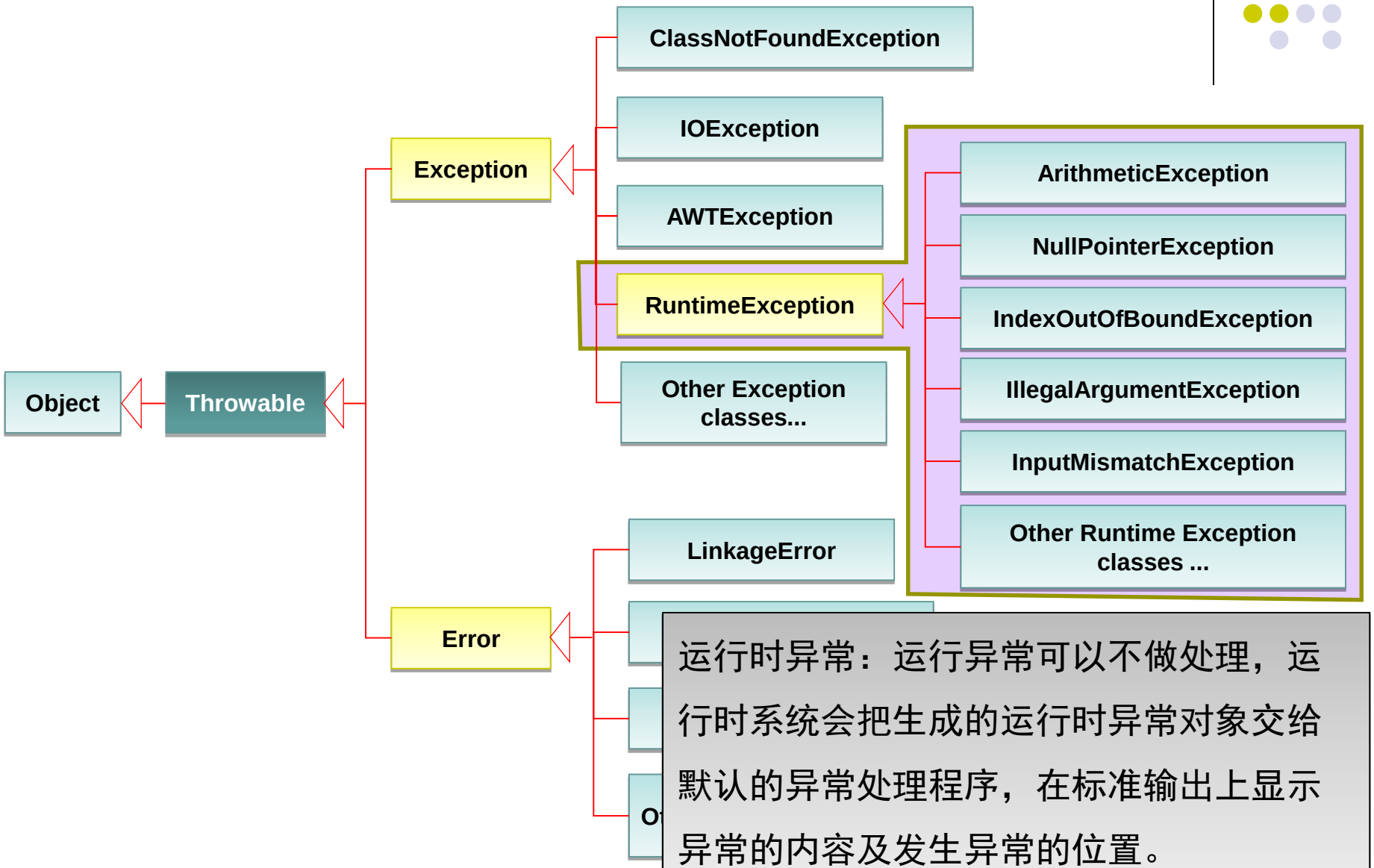
异常类继承结构图——异常

异常（Exception）描述由程序和外部环境引起的错误，这些错误能够通过程序捕获和处理。



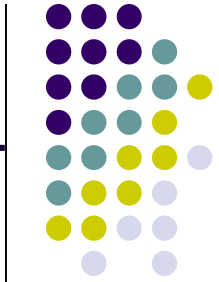
运行时异常（Runtime Exception）





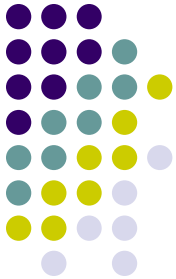
运行时异常：运行异常可以不做处理，运行时系统会把生成的运行时异常对象交给默认的异常处理程序，在标准输出上显示异常的内容及发生异常的位置。

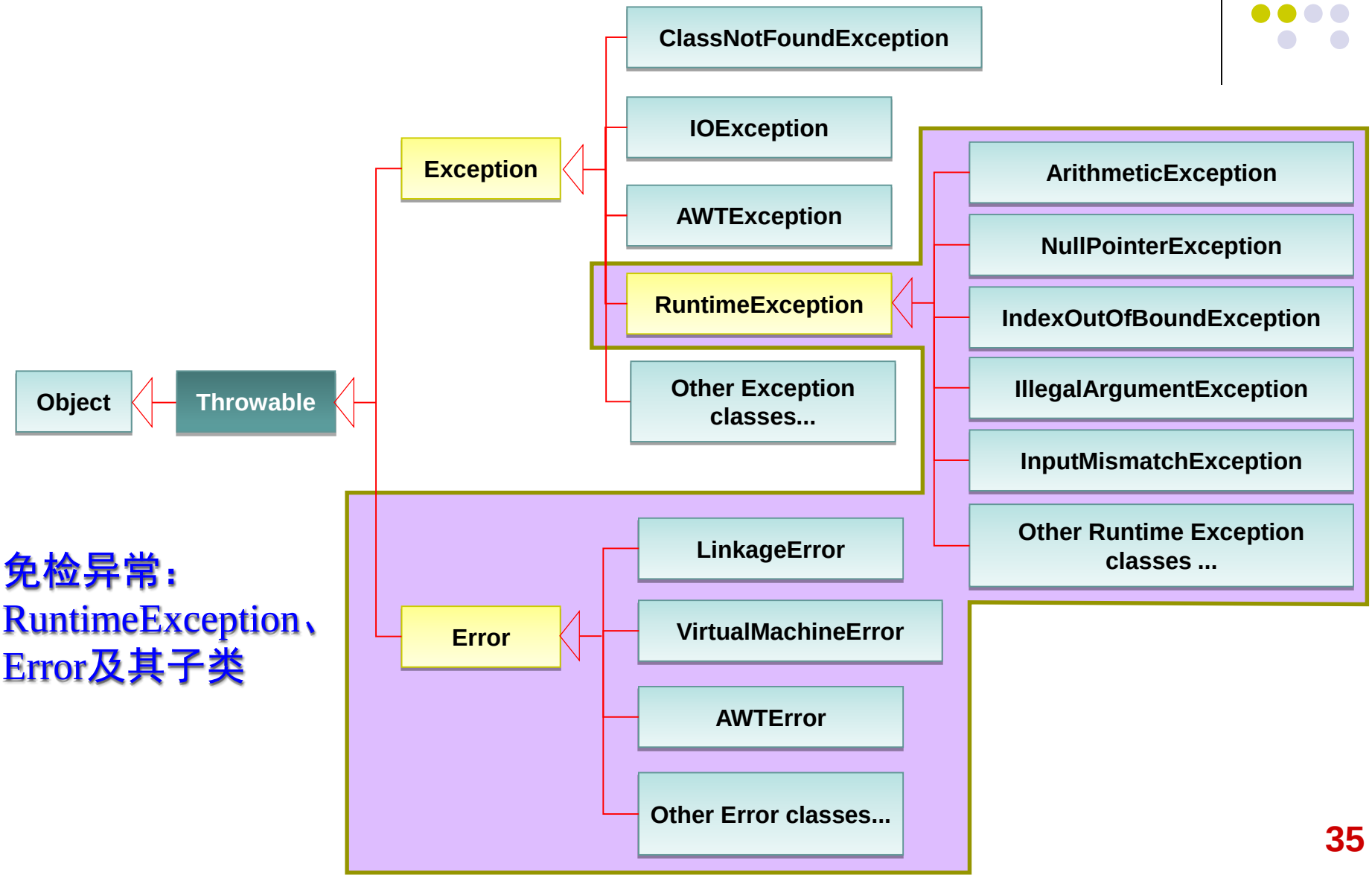
5.2 Java的异常处理机制



- 免检异常(Unchecked Exception):
  - RuntimeException、Error以及它们的子类都称为免检异常（unchecked exception）。
- 必检异常(Checked Exception):
  - 所有其他异常都称为必检异常（checked exception），意思是指编译器会强制程序员检查并处理它们。

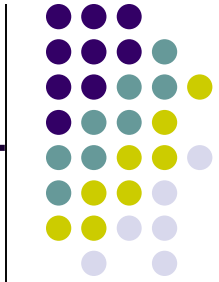
5.2 Java的异常处理机制





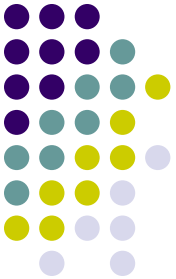
免检异常：
RuntimeException、
Error及其子类

5.2 Java的异常处理机制



- 免检异常
- 大多数情况下，免检异常反映程序设计中不可重获的逻辑错误。
  - 例如，通过一个未赋值的引用变量访问一个对象，会抛出NullPointerException异常，
  - 越界访问一个数组的元素就会抛出IndexOutOfBoundsException异常，这些都是程序中必须纠正的逻辑错误。
  - 免检异常可能在程序任何地方出现。
  - 为避免过多地使用try-catch块，Java语言不建议编写捕获或声明免检异常的代码。

5.2 Java的异常处理机制

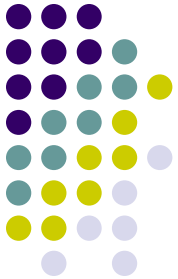


- 奇怪的现象

```
public class TestThrows {
    public static void main(String[] args) {
            FileInputStream fis = new FileInputStream("a.txt");
    }
}
```

为什么“完全正确”的代码不能编译？

5.2 Java的异常处理机制



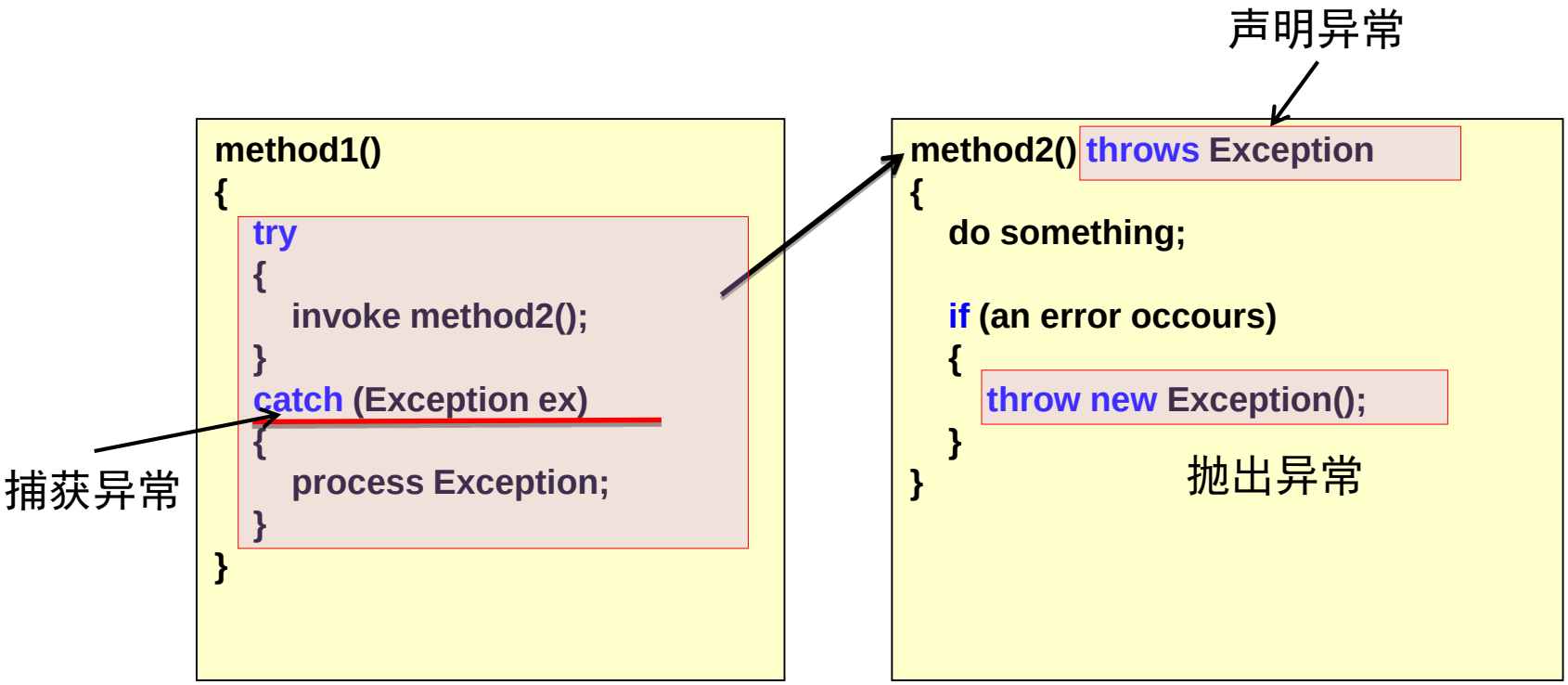
- 修正错误

```
public class TestThrows{
    public static void main(String[] args)
            throws FileNotFoundException
    {
            FileInputStream fis = new FileInputStream("a.txt");
    }
}
```

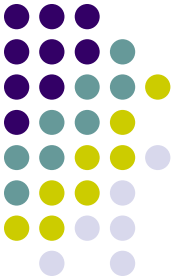
必检异常

5.2 Java的异常处理机制

- 声明、抛出和捕获异常



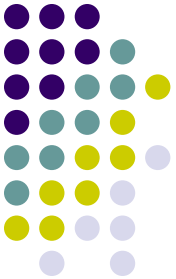
5.2 使用Java异常处理机制



throws语句

- throws语句表明某方法中可能出现某种（或多种）异常，但它自己不能处理这些异常，而由调用者来处理。
- 当一个方法包含throws子句时，需要在调用此方法的代码中使用try/catch/finally进行捕获，或者是重新对其进行声明，否则编译时报错。

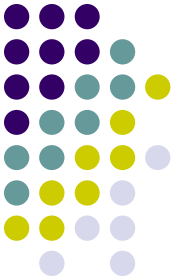
5.2 Java的异常处理机制



- 声明异常
- 每个方法都必须说明它可能抛出的必检异常的类型，这称为声明异常（*declaring exception*）
- 声明异常时，使用关键字：throws

```
// 这里声明的都是“必检异常”
public void readFile(String filename) throws FileNotFoundException
{
   // ... ...
}

// 可以声明多个异常，用逗号分开
// 也可以声明自定义异常，如 MyException
public void writeFile() throws IOException, MyException
{
   // ... ...
}
```

5.2 使用Java异常处理机制



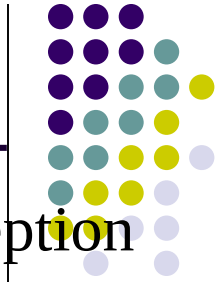
抛出多个受控异常的方法

- 声明抛出多个异常的方法

  int g(float h) throws OneException,TwoException { …… }

- 测试示例:ThrowMultiExceptionsDemo.java
- 注意一个奇特的地方：
- 当一个方法声明抛出多个异常时，在此方法调用语句处只要catch其中任何一个异常，代码就可以顺利编译。

5.2 Java的异常处理机制

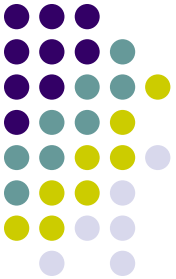


- throws语句中声明的异常称为必检异常（checked exception），通常直接派生自Exception类。

```
// 可以声明多个异常，用逗号分开
// 也可以声明自定义异常，如 MyException
public void writeFile() throws IOException, MyException
{
   // ... ...
}
```

- 注意：当一个方法声明抛出多个异常时，在此方法调用语句处只要catch其中任何一个异常，代码就可以顺利编译。

抛出异常



- 程序检查到一个错误后，创建一个适当异常类型的实例并抛出它，这就称为抛出异常（*throwing exception*）。
- 抛出异常使用关键字 throw

```
public void readFile(String filename) throws FileNotFoundException
{
   if (file "filename" not found)
   {
      // 方法1：
      throw new FileNotFoundException();

      // 方法2：
      FileNotFoundException ex = new FileNotFoundException();
      throw ex;
   }
}
```

抛出异常举例

```
public void setRadius(double newRadius) throws IllegalArgumentException
{
   if (newRadius >= 0)
   {
      radius = newRadius;
   }
   else
   {
      throw new IllegalArgumentException("Radius must >= 0");
   }
}
```

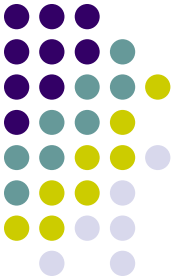
IllegalArgumentException是JDK中定义的一个异常。

它有一个带有字符串参数的构造方法。

Java中，异常的命名方式是：XXXException

注：
IllegalArgumentException是Java中的一个“免检异常”，是不需要进行声明的。
这里为了举例，使用了这个异常。
这种做法是不推荐的。

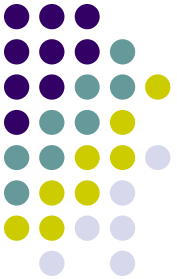
5.2 Java的异常处理机制



- 捕获异常

```
try
{
        statements.   // statements may throw exceptions.
}
catch (Exception1 ex1)
{
    handle for Exception1
}
catch (Exception2 ex2)
{
    handle for Exception2
}
...
catch (ExceptionN exN)
{
    handle for ExceptionN
}
```

5.2 Java的异常处理机制



- 捕获异常
- 异常的“多态”特性
  - 可以有多个catch语句块，每个代码块捕获一种异常。
  - 在某个try块后有两个不同的catch块捕获两个相同类型的异常是语法错误。
  - 使用catch语句，只能捕获Exception类及其子类的对象。因此，一个捕获Exception对象的catch语句块可以捕获所有“可捕获”的异常。
  - 将catch (Exception e)放在别的catch块前面会使这些catch块都不执行，因此，Java不会编译这个程序。

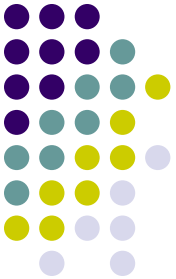
```java
public class CircleWithException {
    private double radius;
    private static int numberOfObjects = 0;

    public CircleWithException() {
        this(1.0);
    }

    public CircleWithException(double newRadius) {
        setRadius(newRadius);
        numberOfObjects++;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double newRadius) throws IllegalArgumentExceptio
        if (newRadius >= 0)
            radius = newRadius;
        else
            throw new IllegalArgumentException("Radius cannot be negative")
    }

    public static int getNumberOfObjects(){
        return numberOfObjects;
    }
}
```

```
public class TestCircleWithException
{
    public static void main(String[] args)
    {
        try
        {
            CircleWithException c1 = new CircleWithException(5);
            CircleWithException c2 = new CircleWithException(-5);
            CircleWithException c3 = new CircleWithException(0);
        }
        catch (IllegalArgumentException ex)
        {
            System.out.println(ex);
        }

        System.out.println("Number of objects created: " +
                           CircleWithException.getNumberOfObjects());
    }
}
```

执行程序后，输出结果为：

java.lang.IllegalArgumentException: Radius cannot be negative
Number of objects created: 1

捕获或声明必检异常



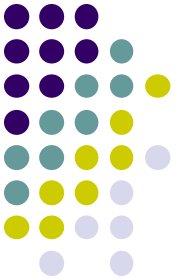
- Java强迫程序员处理必检异常。如果方法声明了一个必检异常（即Error或Runtime-Exception之外的异常），必须:
  - 在try-catch块中捕获它
  - 或者在调用它的方法中再次声明该异常
- 例，假定方法p1调用方法p2，p2可能抛出一个必检异常（比如：IOException）：

```
// 写法一：
// 在 p1 中，使用 try...catch 捕获异常
// 并处理之
void p1()
{
   try
   {
      p2();  // 调用 p2() 这个方法
   }
   catch (IOException ex)
   {
      // process ex
   }
}
```

```
// 写法二：
// 在 p1 中，不捕获异常，再次声明该异常
void p1() throws IOException
{
   p2(); // 调用 p2() 这个方法
}
```

```
// p2 声明了一个 IOException 异常
void p2() throws IOException
{
}
```

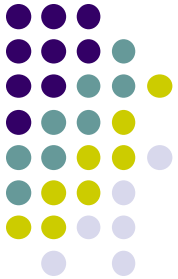
Finally子句



- 不论异常是否出现或者是否被捕获，都希望执行某些代码。Java有一个finally子句可以用来达到这一目的。

```
try
{
    statements;
}
catch(TheException ex)
{
    handling ex;
}
finally
{
    finalStatements;
}
```

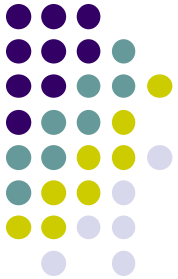
例子1：



假设语句中没有异常

```
try
{
    statements;
}
catch(TheException ex)
{
    handling ex;
}
finally
{
    finalStatements;
}

Next statetments
```

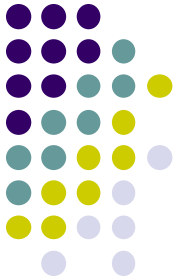
例子1：



```
try
{
     statements;
}
catch(TheException ex)
{
     handling ex;
}
finally
{
     finalStatements;
}

Next statetments
```

finally语句总是被执行

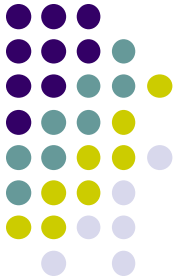
例子1：



```
try
{
     statements;
}
catch(TheException ex)
{
     handling ex;
}
finally
{
     finalStatements;
}

Next statetments
```

继续执行

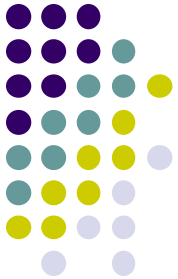
例子2：



```
try
{
    statement1;
    statement2;
    statement3;
}
catch(Exception1 ex)
{
    handling ex;
}
finally
{
    finalStatements;
}

Next statetments
```

statement1中没有异常

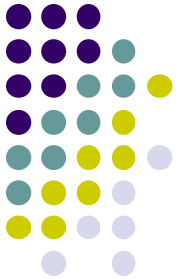
例子2：



```
try
{
    statement1;
    statement2;
    statement3;
}
catch(Exception1 ex)
{
    handling ex;
}
finally
{
    finalStatements;
}

Next statetments
```

假设在statement2中发生了一个Exception1类型的异常

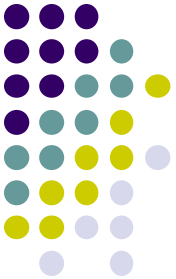
例子2：



```
try
{
     statement1;
     statement2;
     statement3;
}
catch(Exception1 ex)
{
     handling ex;
}
finally
{
     finalStatements;
}

Next statetments
```

处理异常

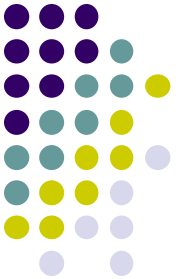
例子2：



```
try
{
     statement1;
     statement2;
     statement3;
}
catch(Exception1 ex)
{
     handling ex;
}
finally
{
     finalStatements;
}

Next statetments
```

finally语句总是被执行

例子2:

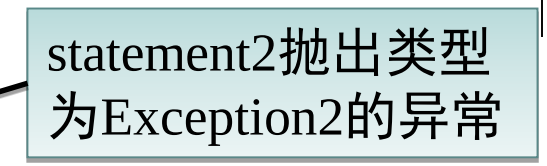


```
try
{
     statement1;
     statement2;
     statement3;
}
catch(Exception1 ex)
{
     handling ex;
}
finally
{
     finalStatements;
}

Next statetments
```

继续执行执行

例子3：

```
try
{
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1)
{
    handling ex1;
}
catch (Exception2 ex2)
{
    handling ex2;
}
finally
{
    finalStatements;
}

Next statetments
```

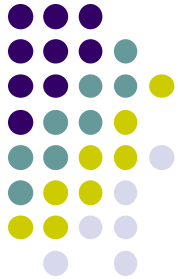


例子3：

```
try
{
     statement1;
     statement2;
     statement3;
}
catch (Exception1 ex1)
{
     handling ex1;
}
catch (Exception2 ex2)
{
     handling ex2;
}
finally
{
     finalStatements;
}

Next statetments
```

处理异常

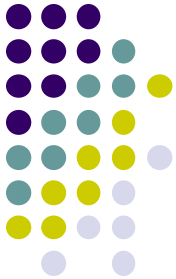
例子3：



```
try
{
     statement1;
     statement2;
     statement3;
}
catch (Exception1 ex1)
{
     handling ex1;
}
catch (Exception2 ex2)
{
     handling ex2;
}
finally
{
     finalStatements;
}

Next statetments
```

finally总是被执行

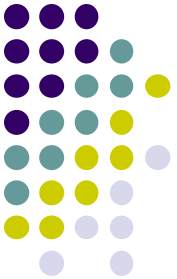
例子3：



```
try
{
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1)
{
    handling ex1;
}
catch (Exception2 ex2)
{
    handling ex2;
}
finally
{
    finalStatements;
}

Next statetments
```

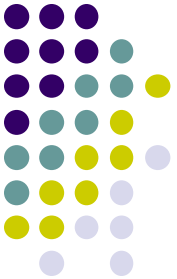
继续执行

5.2 使用Java异常处理机制



- 把可能会发生错误的代码放进try语句块中。
- 当程序检测到出现了一个错误时会抛出一个异常对象。异常处理代码会捕获并处理这个错误。
- catch语句块中的代码用于处理错误。
- 当异常发生时，程序控制流程由try语句块跳转到catch语句块。
- 不管是否有异常发生，finally语句块中的语句始终保证被执行。
- 如果没有提供合适的异常处理代码，JVM将会结束掉整个应用程序。

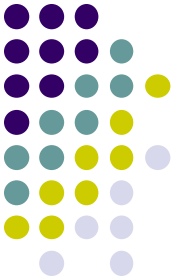
5.2 使用Java异常处理机制



多层次嵌套的异常捕获

- 当存在方法调用时，经常出现“嵌套”的异常捕获结构。
- Demo: CatchWho.java
- 要注意哪个异常由哪个进行捕获
- CatchWho2.java程序运行的结果

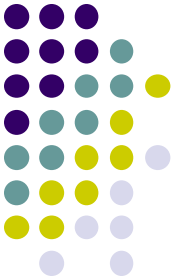
5.2 使用Java异常处理机制



多层次嵌套的finally

- 当有多个嵌套的try…catch…finally时，要特别注意finally的执行时机。
- Demo: EmbedFinally.java
- 特别注意：
- 当有多层嵌套的finally时，异常在不同的层次抛出，在不同的位置抛出，可能会导致不同的finally语句块执行顺序。

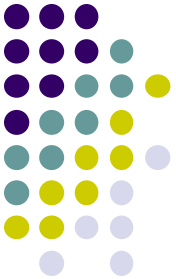
5.2 使用Java异常处理机制



finally语句块一定会执行吗？

- System.exit()方法可能会导致finally语句块不执行。
- Demo: SystemExitAndFianlly.java

5.2 使用Java异常处理机制



跟踪异常的传播路径

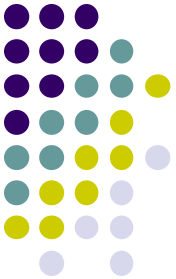
- 当程序中出现异常时，JVM会依据方法调用顺序依次查找有关的错误处理程序。
- 可使用printStackTrace 和 getMessage方法了解异常发生的情况：

1、printStackTrace：打印方法调用堆栈。

2、每个Throwable类的对象都有一个getMessage 方法，它返回一个字串，这个字串是在Exception构造函数中传入的，通常让这一字串包含特定异常的相关信息。

3、Demo: PrintExpressionStack.java

第五章 异常处理

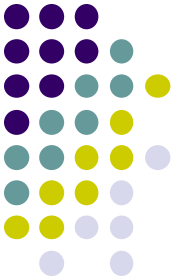


5.1 错误处理的方法概述

5.2 Java的异常处理机制

5.3 创建自己的异常类

5.3 创建自己的异常类



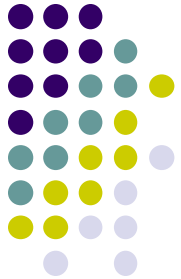
何时使用异常

- 一个方法出现异常时，如果想让该方法的调用者处理异常，应该创建一个异常对象并将其抛出。
- 如果能在发生异常的方法中处理异常，那么就不需要抛出异常

即：自己（本方法）的事情（发生了异常），尽量自己解决。

自己解决不了，才将问题丢出来（抛出异常）。

何时使用异常



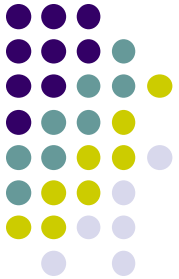
- 在代码中，应该什么时候使用try-catch块呢？
  - 当必须处理不可预料的错误时应该使用它。
  - 不要用try-catch块处理简单的、可预料的情况。例如下述代码：

```
try
{
    System.out.println(ref.toString());
}
catch (NullPointerException npe)
{
    System.out.println("ref is null");
}
```

建议替换为：

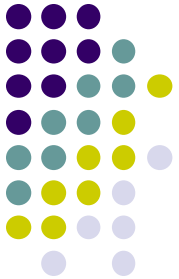
```
if (ref != null)
    System.out.println(ref.toString());
else
    System.out.println("ref is null");
```

创建自定义异常类



- 尽量使用JDK中的异常类。
- 如果预先定义的类不够的话就该创建自定义异常类。
- 扩展Exception或Exception子类来声明自定义异常类。

创建自定义异常类



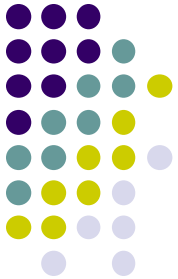
- 继承Exception及其子类，多数情况下为非运行时异常。
- class MyException extends Exception{

  … String say(){…}

  }
- 在程序中使用自己的异常类

```
class UseMyException{
      …
  try
      { throw new MyException();}
  catch(MyException e)
       {System.out.println(e.say());}
}
```

自定义异常类举例



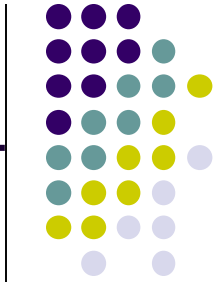
- 创建一个自定义异常类：InvalidRadiusException，表示半径值非法
  - 在Java中，异常类的命名方式一般是：XXXException

```
public class InvalidRadiusException extends Exception
{
    private double radius;

    public InvalidRadiusException(double radius)
    {
        super("Invalid radius " + radius);
        this.radius = radius;
    }
    public double getRadius()
    {
        return radius;
    }
}
```

```
public class CircleWithUserException {
    private double radius;
    private static int numberOfObjects = 0;

    public CircleWithUserException() throws InvalidRadiusException {
        this(1.0);
    }

    public CircleWithUserException(double newRadius) throws InvalidRadiusException
    {
        setRadius(newRadius);
        numberOfObjects++;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double newRadius) throws InvalidRadiusException {
        if (newRadius >= 0)
            radius = newRadius;
        else
            throw new InvalidRadiusException(newRadius);
    }

    public static int getNumberOfObjects(){
        return numberOfObjects;
    }
}
```

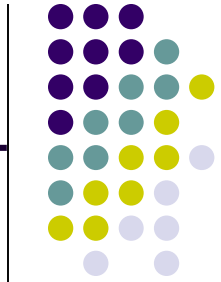
```
public class TestCircleWithUserException
{
    public static void main(String[] args)
    {
        try
        {
            CircleWithUserException c1 = new CircleWithUserException(5);
            CircleWithUserException c2 = new CircleWithUserException(-5);
            CircleWithUserException c3 = new CircleWithUserException(0);
        }
        catch (InvalidRadiusException ex)
        {
            System.out.println("Invalid radius exception occoured,
                                radius = " + ex.getRadius());
        }

        System.out.println("Number of objects created: " +
                           CircleWithUserException.getNumberOfObjects());
    }
}
```

关于开发中异常处理的建议



- 在中间层组件中抛出异常，在界面层组件中捕获异常
- 在底层组件中捕获JVM抛出的“只有程序员能看懂的”异常，转换为中间层的业务逻辑异常，再由界面层捕获以提供有意义的信息。
- 自身能够处理的异常，不要再向外界抛出。
- 尽可能地在靠近异常发生的地方捕获并处理异常。
- 尽可能地捕获最具体的异常类型，不要在中间层catch(Exception)
- 在开发阶段捕获并显示所有异常信息，发布阶段要移除部分代码，以避免“过于专业”的异常信息困扰用户，特别地，系统发布之后，不要将服务端异常的详细信息发给客户端，以免被黑客利用。

小结



- 理解异常和异常处理
- 区别异常的类型：Error（严重的）与Exception（不严重的），必检异常与免检异常
- 在方法头中声明异常
- 在方法中抛出异常
- 用try-catch块处理异常
- 解释异常的传播
- 在try-catch块中重新抛出异常
- 在try-catch块中使用finally子句
- 了解何时使用异常
- 声明自定义异常类