

---

# Announcement

- No class Friday April 9th.
- Last day of lectures: Tuesday, April 13 at the same time and place.

---

# Exception handling

- An *exception* is an object that represents a special situation or error that occurs at *runtime*
- If the error or situation occurs, we say that the exception is *raised* or *thrown*.
- An exception may be thrown by the Java Runtime System (JVM) or explicitly by the program using the `throw` keyword.
- An exception can be handled by the `try-catch` construct.
- An exception handled by the `try-catch` construct is said to be caught.
- Exception objects must be instances of some subclass of `Exception`, or must implement the `Throwable` interface.

---

## Exception handling

- An exception is generated (raised) with the *throw* statement:

```
throw object;
```

where *object* is an instance of a subclass of `Exception` or `Throwable`

- The *try-catch* statement:

```
try {  
    statements;  
}  
catch (ExceptionSubclass1 e) {  
    statements1;  
}  
catch (ExceptionSubclass2 e) {  
    statements2;  
}  
.  
.  
.
```

---

## Exception handling

- A try-catch statement executes its default statements in sequence, and
  - If no exception is raised, then computation continues after the catch clauses
  - Otherwise, if an exception is raised, the sequence of statements is interrupted, and execution continues in the catch clause that matches the type of the exception
- After a catch clause finishes, computation continues after the try-catch. This is, the flow of control does not return to the point where the exception occurred. *Note: It never returns to the method that raised the exception, in contrast with a method call.*
- An exception which is not caught by a try-catch, is “propagated”, i.e. it is raised again

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    {
        int a, c, d;
        a = 5;
        c = a / b;
        d = c + 2;
        return d;
    }
    static void client()
    {
        int x = 5, y;
        y = some_method(x);
        System.out.println(y);
    }
}
```

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    {
        int a, c, d;
        a = 5;
        if (b != 0) {
            c = a / b;
            d = c + 2;
        }
        else {
            d = 3;
        }
        return d;
    }
    static void client()
    {
        int x = 5, y;
        y = some_method(x);
        System.out.println(y);
    }
}
```

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    {
        int a, c, d;
        try {
            a = 5;
            c = a / b;
            d = c + 2;
        }
        catch (ArithmeticException e) {
            d = 3;
        }
        return d;
    }
    static void client()
    {
        int x = 5, y;
        y = some_method(x);
        System.out.println(y);
    }
}
```

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    {
        int a, c, d;
        try {
            a = 5;
            c = a / b;
            d = c + 2;
        }
        catch (ArithmeticException e) {
            d = 3;
        }
        return d;
    }
    static void client()
    {
        int x = 0, y;
        y = some_method(x);
        System.out.println(y);
    }
}
```



---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    {
        int a, c, d;
        a = 5;
        c = a / b;
        d = c + 2;
        return d;
    }
    static void client()
    {
        int x = 5, y;
        try {
            y = some_method(x);
        }
        catch (ArithmeticException e) {
            y = 3;
        }
        System.out.println(y);
    }
}
```

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    throws ArithmeticException
    {
        int a, c, d;
        a = 5;
        c = a / b;
        d = c + 2;
        return d;
    }
    static void client()
    {
        int x = 5, y;
        try {
            y = some_method(x);
        }
        catch (ArithmeticException e) {
            y = 3;
        }
        System.out.println(y);
    }
}
```

---

## Exception handling

```
class SomeClass {
    static int some_method(int b)
    throws ArithmeticException
    {
        int a, c, d;
        a = 5;
        c = a / b;
        d = c + 2;
        return d;
    }
    static void client()
    {
        int x = 0, y;
        try {
            y = some_method(x);
        }
        catch (ArithmeticException e) {
            y = 3;
        }
        System.out.println(y);
    }
}
```

---

---

## Exception handling

```
class Creature {  
    void move()  
    {  
        System.out.println("Here we go...");  
    }  
}
```

```
class Zoo {  
    void animate(Creature c)  
    {  
        c.move();  
    }  
}
```

---

## Exception handling

```
public class ZooTest {
    public static void main(String[] args)
    {
        Zoo my_zoo = new Zoo();
        Creature argos;
        my_zoo.animate(argos);
    }
}
```

---

## Exception handling

```
public class ZooTest {
    public static void main(String[] args)
    {
        Zoo my_zoo = new Zoo();
        Creature argos;
        my_zoo.animate(argos); // Null-pointer except
    }
}
```

---

## Exception handling

```
class Creature {
    void move()
    {
        System.out.println("Here we go...");
    }
}

class Zoo {
    void animate(Creature c)
    {
        if (c != null) c.move();
        else System.out.println("Invalid access");
    }
}
```

---

## Exception handling

```
public class ZooTest {
    public static void main(String[] args)
    {
        Zoo my_zoo = new Zoo();
        Creature argos = new Creature();
        my_zoo.animate(argos);
    }
}
```



---

## Exception handling

```
class Creature {
    void move()
    {
        System.out.println("Here we go...");
    }
}

class Zoo {
    void animate(Creature c)
    {
        try {
            c.move();
        }
        catch (NullPointerException e) {
            System.out.println("Invalid access");
        }
    }
}
```

---

## Exception handling

```
class Creature {  
    void move()  
    {  
        System.out.println("Here we go...");  
    }  
}
```

```
class Zoo {  
    void animate(Creature c)  
    throws NullPointerException  
    {  
        c.move();  
    }  
}
```

---

## Exception handling

```
public class ZooTest {
    public static void main(String[] args)
    {
        try {
            Zoo my_zoo = new Zoo();
            Creature argos;
            my_zoo.animate(argos);
        }
        catch (NullPointerException e) {
            System.out.println("Invalid access");
        }
    }
}
```

---

## Exception handling

```
static int q(float f) {
    if (f < 10)
        System.out.println("Error, f <10, "+f);
    return f * 3 + 1;
}
static void p() {
    float n = Keyboard.readFloat();
    int m = q(n);
    System.out.println(m);
}
```

---

## Exception handling

```
static int q(float f) {
    if (f < 10) {
        System.out.println("Error, f <10, "+f);
        return -1;
    }
    return f * 3 + 1;
}

static void p() {
    float n = Keyboard.readFloat();
    int m = q(n);
    if (m != -1)
        System.out.println(m);
    else
        System.out.println("Error");
}
```

---

## Exception handling

```
static int q(float f) {
    if (f < 10) return -1;
    return f * 3 + 1;
}
static void p() {
    float n = Keyboard.readFloat();
    int m = q(n);
    if (m != -1)
        System.out.println(m);
    else
        System.out.println("Error"); //No Error info
}
```

---

## Exception handling

```
static int q(float f) {
    if (f < -5) return -1;
    return f * 3 + 1;
}
static float r(float f) {
    if (f > 15) return -1;
    return f - 2;
}
static void p() {
    float n = Keyboard.readFloat();
    int m = q(r(n));
    if (m != -1)
        System.out.println(m);
    else
        System.out.println("Error"); //No Error info
}
// q(r(13)) = q(11) = 34
// q(r(16)) = q(-1) = -2 // wrong
// q(r(-6)) = q(-8) = -1
```

---

## Exception handling

```
static int q(float f) {
    if (f < -5) return -1;
    return f * 3 + 1;
}
static float r(float f) {
    if (f > 15) return -1;
    return f - 2;
}
static void p() {
    float n = Keyboard.readFloat();
    int partial1 = r(n);
    if (partial1 == -1)
        System.out.println("Error in r");
    else {
        int partial2 = q(partial1);
        if (partial2 == -1)
            System.out.println("Error in q"); //No Error
        else
            System.out.println(partial2);
    }
}
```

---



---

## Exception handling

```
class MyException extends Exception {
    String message;
    MyException(String m)
    {
        message = m
    }
    public String toString()
    {
        return "MyException occurred: "+message;
    }
}
```

---

## Exception handling

```
static int q(float f) throws MyException
{
    if (f < -5)
        throw new MyException("q: "+f);
    return f * 3 + 1;
}
static float r(float f) throws MyException
{
    if (f > 15)
        throw new MyException("r: "+f);
    return f - 2;
}
```

---

## Exception handling

```
static void p() {  
    float n = Keyboard.readFloat();  
    try {  
        int m = q(r(n));  
        System.out.println(m);  
    }  
    catch (MyException e) {  
        System.out.println(e);  
    }  
}
```

---

## Exception handling

```
class MyException extends Exception {
    String message;
    MyException(String m) { message = m; }
    public String toString()
    {
        return "MyException occurred: "+message;
    }
}

class MyOtherException extends Exception {
    int code;
    MyOtherException(int c) { code = c; }
    public String toString() { return ""+code; }
}
```

---

## Exception handling

```
static int q(float f) throws MyOtherException
{
    if (f < -5)
        throw new MyOtherException(7);
    return f * 3 + 1;
}
static float r(float f) throws MyException
{
    if (f > 15)
        throw new MyException("r: "+f);
    return f - 2;
}
```

---

## Exception handling

```
static void p()
{
    float n = Keyboard.readFloat();
    try {
        int m = q(r(n));
        System.out.println(m);
    }
    catch (MyException e) {
        System.out.println(e);
    }
    catch (MyOtherException e) {
        String s = e;
    }
}
```

---

## Exception handling

- If MyOtherException is not caught, it repropagates

```
static void p() throws MyOtherException
{
    float n = Keyboard.readFloat();
    try {
        int m = q(r(n));
        System.out.println(m);
    }
    catch (MyException e) {
        System.out.println(e);
    }
}
```

- Note: p does not throw an exception explicitly

---

## Exception handling

```
class Food {
    boolean fresh, smelly;
}
class FoulSmell extends Exception {
    public String toString() {
        return "Yuck";
    }
}
class FoodPoison extends Exception {
    public String toString() {
        return "Ouch";
    }
}
```



---

## Exception handling

```
static void smell(Food f) throws FoulSmell
{
    if (f.smelly)
        throw new FoulSmell();
    System.out.println("Smells OK");
}
static void eat(Food f) throws FoodPoison
{
    System.out.println("Hmmm...");
    digest(f);
}
static void digest(Food f) throws FoodPoison
{
    if (!f.fresh)
        throw new FoodPoison();
}
```

---

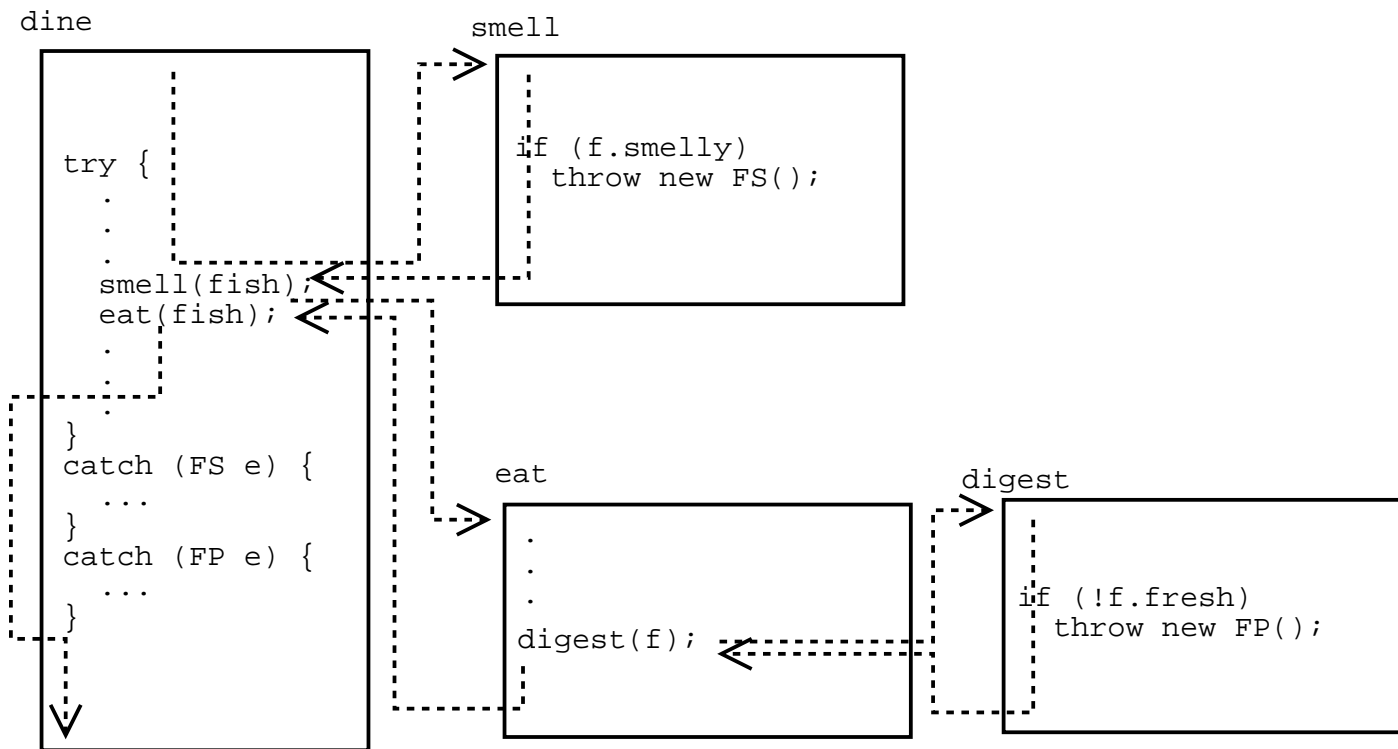
## Exception handling

```
static void dine()
{
    try {
        Food fish = new Food();
        fish.smelly = false;
        fish.fresh = false;
        smell(fish);
        eat(fish);
    }
    catch (FoulSmell e) {
        System.out.println(e);
    }
    catch (FoodPoison e) {
        System.out.println(e);
    }
}
```

---

# Exception handling

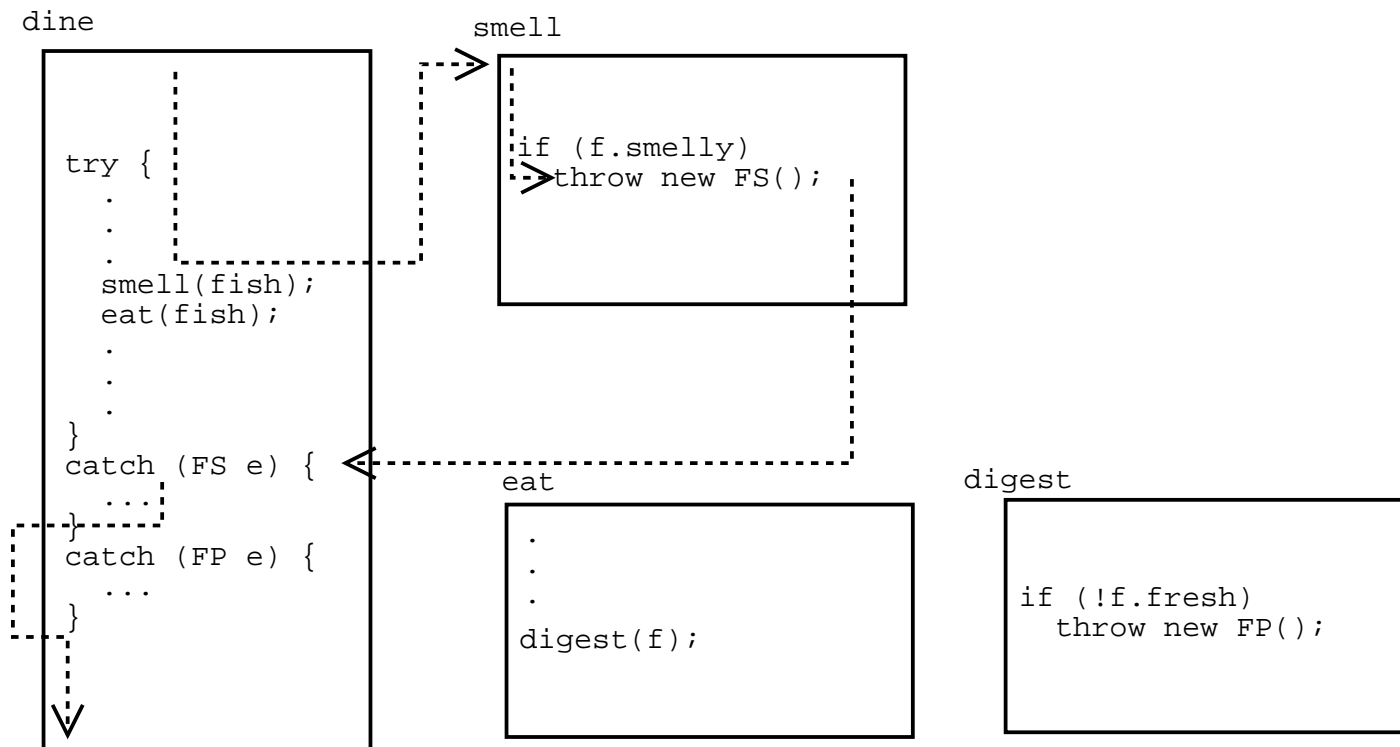
```
// fish.smelly = false; fish.fresh = true;
```



---

# Exception handling

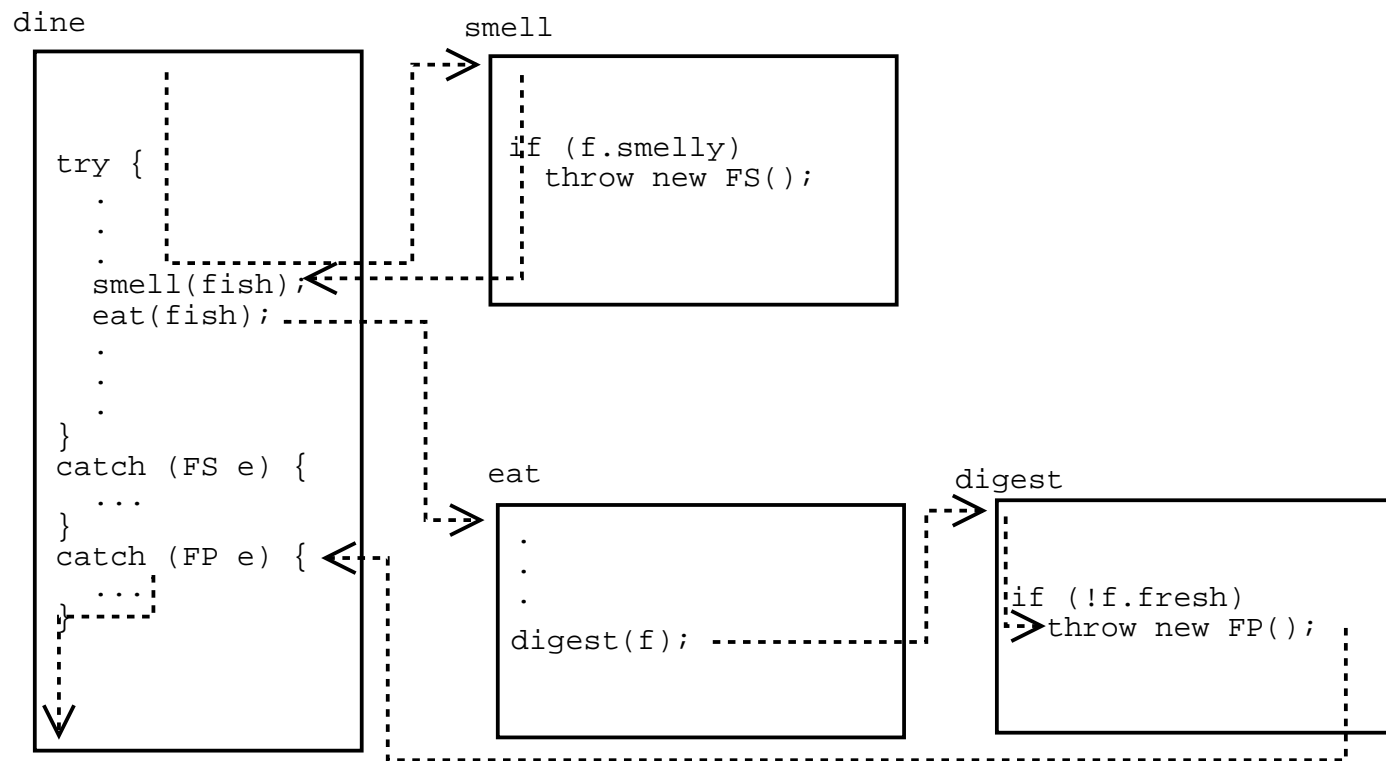
```
// fish.smelly = true;
```



---

# Exception handling

```
// fish.smelly = false; fish.fresh = false;
```



---

## Exception handling

- A method can throw more than one class of exceptions:

```
void m() throws A, B, ...  
{  
    ... throw new A() ...  
    ... throw new B(...) ...  
}
```

- ... but the exception needs not be raised explicitly in the method itself: it can be raised by another method called by m.

---

## Exception handling

- Exceptions can be used not only for errors, but for control-flow too:

```
class Sheep {
    private int id;
    public Sheep(int i) { n = i; }
    public void jump()
    {
        System.out.println("Sheep #" + id + " jumped");
        if (id == 6)
            throw new LoudSound(i);
    }
}
```

---

## Exception handling

```
class LoudSound extends Throwable {
    private int n;
    public LoudSound(int i) { n = i; }
    public toString()
    {
        return "I was in sheep #" + n;
    }
}
```



---

## Exception handling

```
class GoToSleep {
    public static void main(String[] args)
    {
        try {
            for (int i = 1; i < 100; i++) {
                Sheep s = new Sheep(i);
                s.jump();
            }
            System.out.println("zzzz...");
        }
        catch (LoudSound s) {
            System.out.println(s);
        }
    }
}
```

---

# Exception handling

- Some exceptions arise without an explicit throw.
- Some standard exceptions

Exception

  RuntimeException

    IndexOutOfBoundsException

    StringIndexOutOfBoundsException

    ArithmeticException (e.g. division by 0)

    NullPointerException

  NoSuchMethodException

  ClassNotFoundException

---

The end