

---

# Announcement

- Last day of lectures: Tuesday, April 13 at the same time and place.

---

## Abstract classes

- A class with default behaviour:

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

- An abstract class: subclasses must provide implementation

```
abstract class Creature {
    boolean alive;
    abstract void move();
}
```

---

# Interfaces

- Interfaces are (equivalent to) purely abstract classes, i.e. classes where all the methods are abstract

```
interface Creature
{
    void move();
    void eat();
}
```

is the same as

```
abstract class Creature
{
    abstract void move();
    abstract void eat();
}
```

---

# Interfaces

```
class Human implements Creature
{
    void move()
    {
        System.out.println("I'm walking...");
    }
    void eat()
    {
        System.out.println("I'm eating...");
    }
    void jump()
    {
        System.out.println("Up and down...");
    }
}
```

---

## Abstract classes

```
abstract class MusicPlayer {  
    boolean stopped;  
    void play() { stopped = false; }  
    void ff() { }  
    void pause() { stopped = true; }  
    abstract void stop();  
}
```

---

# Interfaces

- Multiple inheritance is supported for interfaces

```
class A extends B implements C, D, E { ... }
```

- ...because the methods in the interfaces are abstract, which means that they must be implemented in A, so there is no ambiguity problem when calling a method.

---

## Changing visibility in subclasses

- A public method cannot be overridden by a private or protected method:

```
class A {
    public void m()
    {
        System.out.println("A");
    }
}
class B extends A {
    private void m()
    {
        System.out.println("B");
    }
}
```

---

## Changing visibility in subclasses

- A method can be overridden by method with weaker access privileges:

```
class A {
    protected void m()
    {
        System.out.println("A");
    }
}
class B extends A {
    public void m()
    {
        System.out.println("B");
    }
}
```



---

# Object

- Object is a class in the standard Java library which is a superclass to all.
- It contains methods

```
public boolean equals(Object o)
protected Object clone()
public String toString()
public Class getClass()
```

- A method whose argument is of type Object can receive any object from any class as argument. (maximum possible polymorphism.)
- Whenever an object appears in a String expression, the method toString is invoked automatically

---

# Object

```
class Human {
    String name;
    public String toString()
    {
        return "My name is "+name;
    }
}
class Test {
    public static void main(String[] args)
    {
        Human h = new Human();
        h.name = "Kelly";
        String s = ""+h;
        // Same as String s = ""+h.toString();
    }
}
```

---

## Using the Object class

```
import java.util.Vector;
class Test {
    void p() {
        Vector v = new Vector();
        v.addElement(new Integer(2));
        v.addElement(new Integer(5));
        v.insertElementAt(new Integer(3), 1);
        Integer i = (Integer)v.elementAt(2);
        int n = i.intValue();
    }
}
```

---

## Remarks on constructors

- If a class A does not have a constructor, then it implicitly has a default constructor with no parameters

```
A() { super(); }
```

- If a class A has a constructor with parameters, then it does not implicitly have a default constructor A()
- Constructors are not inherited
- All constructors have an implicit call to the superclass's default constructor, unless it explicitly calls a non-default constructor from the parent.
- In a constructor, a call to a parent's constructor must be first.

---

## Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "hello "+q;
    }
}
```

```
public class ConstTest {
    public static void main(String[] args)
    {
        A x = new A(); // Error
        System.out.println(x.s);
    }
}
```

---

## Remarks on constructors

```
class A {  
    String s;  
    A(String q)  
    {  
        s = "hello "+q;  
    }  
}
```

```
public class Constest {  
    public static void main(String[] args)  
    {  
        A x = new A("bye");  
        System.out.println(x.s);  
    }  
}
```

---

## Remarks on constructors

```
class A {
    String s;
    A() { s = "bonjour "; }
    A(String q)
    {
        s = "hello "+q;
    }
}
```

```
public class Constest {
    public static void main(String[] args)
    {
        A x = new A();
        System.out.println(x.s);
    }
}
```

---

## Remarks on constructors

```
class A {
    String s;
    A()
    {
        s = "hello ";
    }
}
class B extends A {
    int n;
}
public class Constest {
    public static void main(String[] args)
    {
        B b1 = new B();
        System.out.println(b1.s);
    }
}
```



---

## Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "ask "+q;
    }
}
class B extends A {
    int n;
}
public class Constest
{
    public static void main(String[] args)
    {
        B b1 = new B();
        System.out.println(b1.s);
    }
}
```

---

## Remarks on constructors

```
class A {  
    String s;  
    A() { s = "hello "; }  
}
```

```
class B extends A {  
    int n;  
    B(int i)  
    {  
        n = i;  
    }  
}
```

```
public class ConstTest {  
    public static void main(String[] args)  
    {  
        B b1 = new B(5);  
        System.out.println(b1.s);  
    }  
}
```

---

## Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {    // Error: no A()
        n = i;
    }
}
public class ConstTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

---

## Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super("bye");
        n = i;
    }
}
public class ConstTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

---

## Remarks on constructors

```
class A {
    String s;
    A() { s = "bye "; }
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super("salut");
        n = i;
    }
}
public class ConstTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

---

# Exception handling

- Errors:
  - Compile-time errors:
    - \* Syntax errors
    - \* Typing errors
  - Run-time errors:
    - \* Logic errors
    - \* Program crashes

---

# Exception handling

```
int some_method()
{
    int a, b, c, d;
    a = 5;
    b = 0;
    c = a / b;
    d = c + 2;
    return d;
}
```

---

## Exception handling

```
int some_method()
{
    int a, b, c, d;
    a = 5;
    b = 0;
    c = a / b; // Run-time exception: div by 0
    d = c + 2;
    return d;
}
// ArithmeticException
```



---

## Exception handling

```
int some_method()
{
    int a, b, c, d;
    a = 5;
    b = Keyboard.readInt();
    c = a / b; // May produce Run-time exception: d
    d = c + 2;
    return d;
}
```

---

## Exception handling

```
int some_method(int b)
{
    int a, c, d;
    a = 5;
    c = a / b; // May produce Run-time exception: d
    d = c + 2;
    return d;
}
```

---

## Exception handling

```
String some_other_method()
{
    int i;
    String s1 = "hello", s2;
    char c;
    i = 5;
    c = s1.charAt(i);
    s2 = "the letter is " + c;
    return s2;
}
// StringIndexOutOfBoundsException
```

---

## Exception handling

```
String some_other_method(int i)
{
    String s1 = "hello", s2;
    char c;
    c = s1.charAt(i);
    s2 = "the letter is " + c;
    return s2;
}
```

---

## Exception handling

```
String some_other_method(int i, String s)
{
    String s2;
    char c;
    c = s.charAt(i);
    s2 = "the letter is " + c;
    return s2;
}
```

---

## 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();  
    }  
}
```

---

## 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

```
String some_other_method(int i, String s)
{
    String s2;
    char c;
    c = s.charAt(i);
    s2 = "the letter is " + c;
    return s2;
}
```

---

## Exception handling

```
String some_other_method(int i, String s)
{
    String s2;
    char c;
    if (s != null && i < s.length()) {
        c = s.charAt(i);
    }
    s2 = "the letter is " + c;
    return s2;
}
```

---

## Exception handling

```
int some_method(int b)
{
    int a, c, d;
    a = 5;
    c = a / b; // May produce Run-time exception: d
    d = c + 2;
    return d;
}
```

---

# Exception handling

```
int some_method(int b)
{
    int a, c, d;
    a = 5;
    if (b != 0) c = a / b;
    else c = 0;
    d = c + 2;
    return d;
}
```

---

## 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

```
int some_method(int b)
{
    int a, c, d;
    a = 5;
    if (b != 0) c = a / b;
    else c = 0;
    d = c + 2;
    return d;
}
```

---

## Exception handling

```
int some_method(int b)
{
    int a, c, d;
    try {
        a = 5;
        c = a / b;
        d = c + 2;
    }
    catch (ArithmeticException e) {
        d = 2;
    }
    return d;
}
```

---

## Exception handling

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

```
class Zoo {  
    void animate(Creature c)  
    {  
        if (c != null) c.move();  
    }  
}
```

---

## Exception handling

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

```
class Zoo {
    void animate(Creature c)
    {
        try {
            c.move();
        }
        catch (NullPointerException e) {
        }
    }
}
```

---

## 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 = 2;
        }
        return d;
    }
    static void yet_another_method()
    {
        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 = 2;
        }
        return d;
    }
    static void yet_another_method()
    {
        int x = 0, y;
        y = some_method(x);
        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 yet_another_method()
    {
        int x = 5, y;
        try {
            y = some_method(x);
        }
        catch (ArithmeticException e) {
            y = 2;
        }
        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 yet_another_method()
    {
        int x = 0, y;
        try {
            y = some_method(x);
        }
        catch (ArithmeticException e) {
            y = 2;
        }
        System.out.println(y);
    }
}
```

---

## Exception handling

- Separation of concerns

```
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

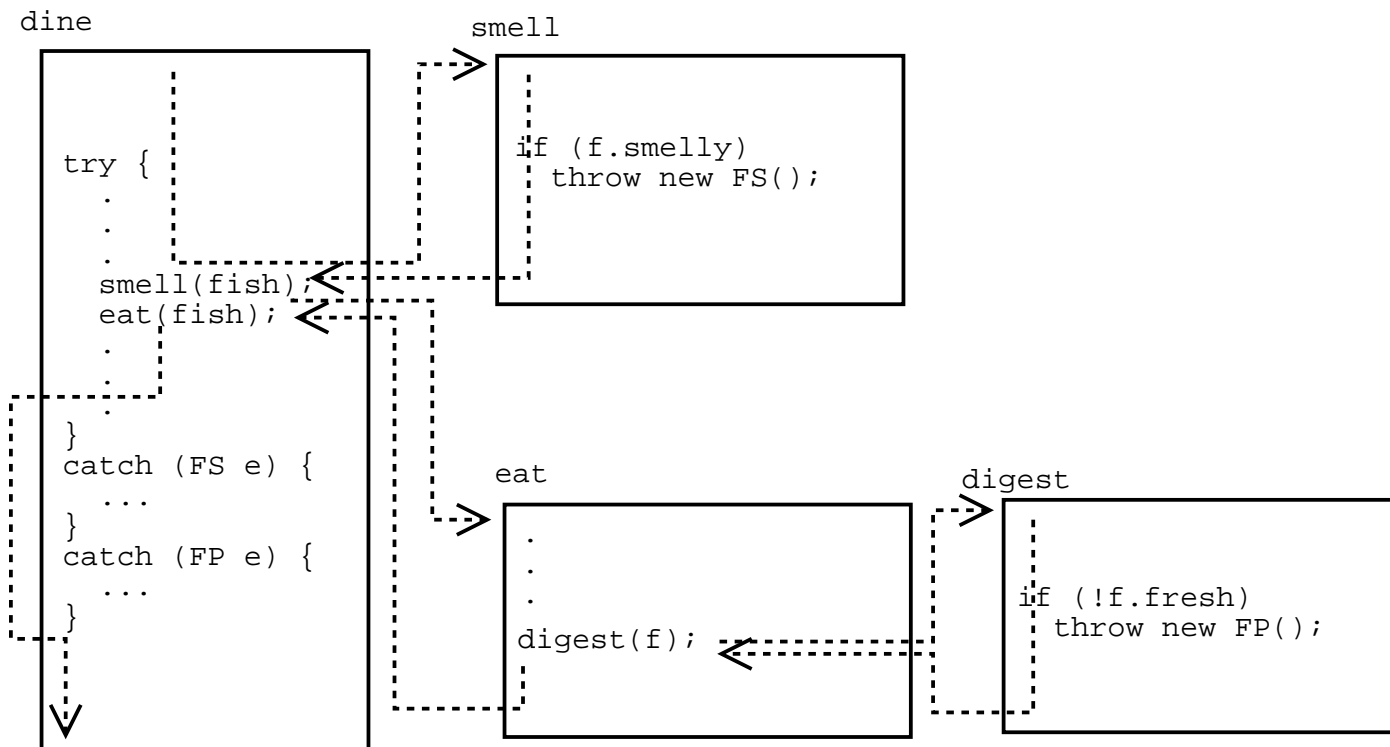
```
static void dine()
{
    try {
        Food fish = new Food();
        fish.smelly = false;
        fish.fresh = false;
        smell(fish);
        eat(fish);
    }
    catch (FoulSmell e) {
        System.out.println(e);
    }
    catch (FoodPoison e) {
        System.out.println(e);
    }
}
```



---

# Exception handling

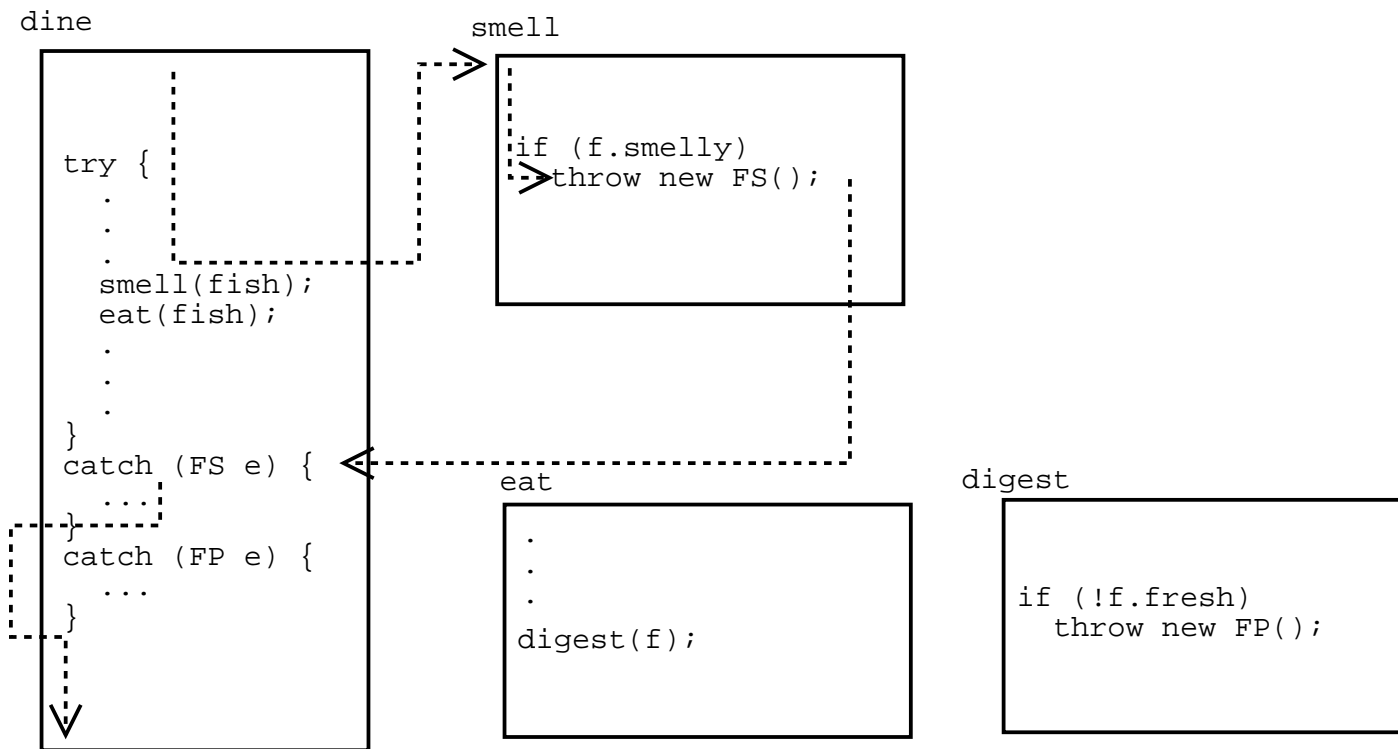
```
// fish.smelly = false; fish.fresh = true;
```



---

# Exception handling

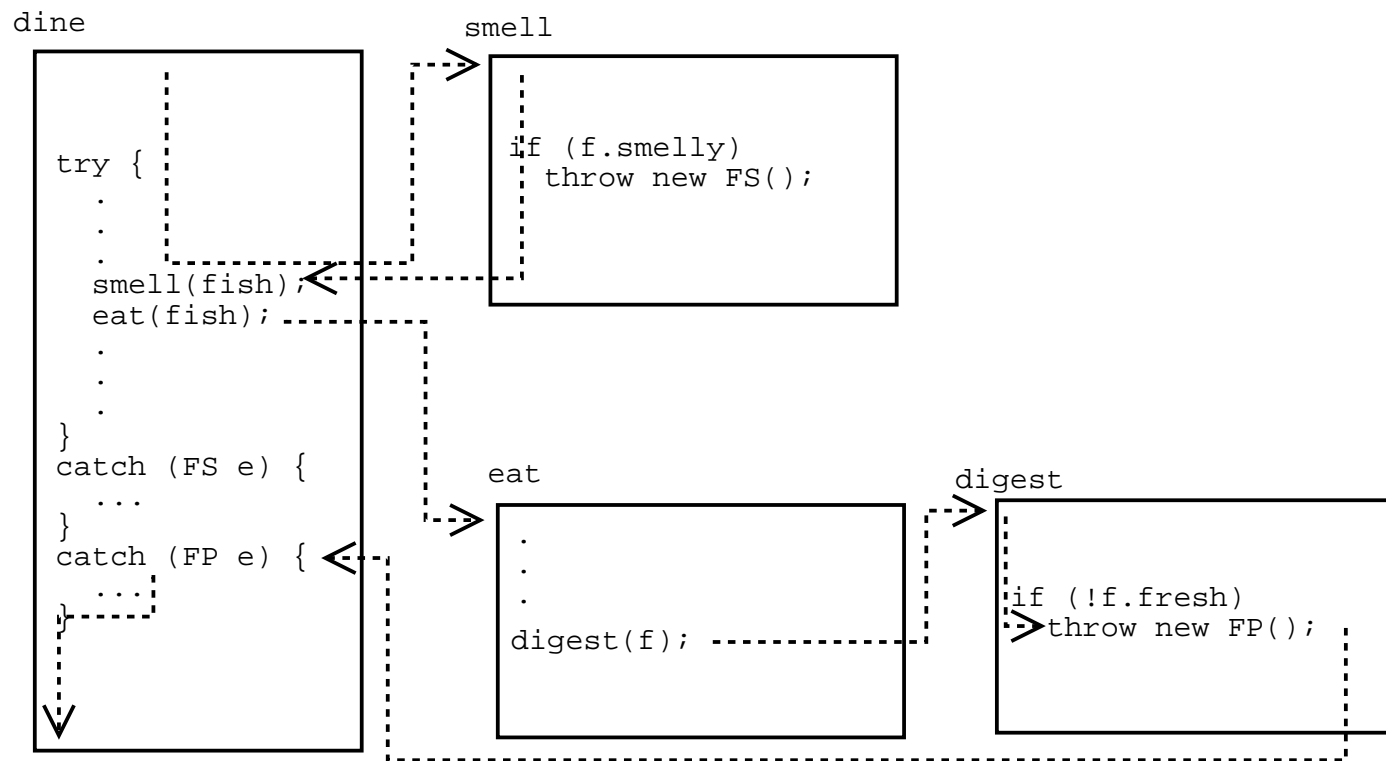
```
// fish.smelly = true;
```



---

# 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