SWINBURNE UNIVERSITY OF TECHNOLOGY

OOP

Exceptions

Content

- Exceptions, what are they? (Objects)
- Generating an Exception (Throwing)
- Dealing with Exceptions (Catching)
- Different types of Exceptions (The Hierarchy)
- Checked v UnChecked Exceptions
- Writing your own Exception
- Designing for Exceptions
  - anticipation and management

Learning Objectives

- Students should understand and know:
  - when & why exceptions are thrown
  - how to deal with exceptions in code using
    - Throws clause, try catch blocks etc.
  - how to write their own exceptions class
  - how exceptions provide structured error handling

Why do programs fail

- Programmers fault:
  - Incorrect implementation of program specification
  - Variables that have not been assigned an object
  - Variables with incorrectly assigned values
  - Index out of bound
  - Inconsistent or inappropriate object state
  - Path or file not found
  - Infinite loops
  - Divide by zero errors……. & …….. the list is endless
- Programmers may get it right, but faults still occur

Even if programmers get it right…

- Abnormal events often arise from the environment:
  - Incorrect URL entered
  - Network interruption
- File processing is particularly error-prone:
  - Missing files
  - Lack of appropriate permissions

How should I manage abnormal events…

- For the program / class I have just written
  - Anticipating Errors
    - Where are errors likely to occur
    - What is the nature of those errors
  - Error handling
    - Can I detect the error
    - What should I do if the error occurs
  - Error reporting
    - Who should I tell, what should I say how should I log the error?

Ways of dealing with problems

Possible responses

- "crash" i.e. stop running
- ignore the problem and keep running
  - maybe cause errors later
  - no good for safety-critical or mission-critical software
- go into some error recovery mode and recover
  - the exception idea- ***catch and handle errors!***

Exceptions

- Provides a structured approach to deal with abnormal events that occur in a program
- **When a code block encounters an abnormal condition (an exception condition) that it can't handle itself...**
  - An Exception is created
  - The exception is thrown
  - Normal program flow-of-control stops
  - The program searches for code that will catch the exception (deal with the exception)
    - Abandons each method on the stack until a handler is found
  - After the exception is dealt with normal program flow starts again

Exception: Definition

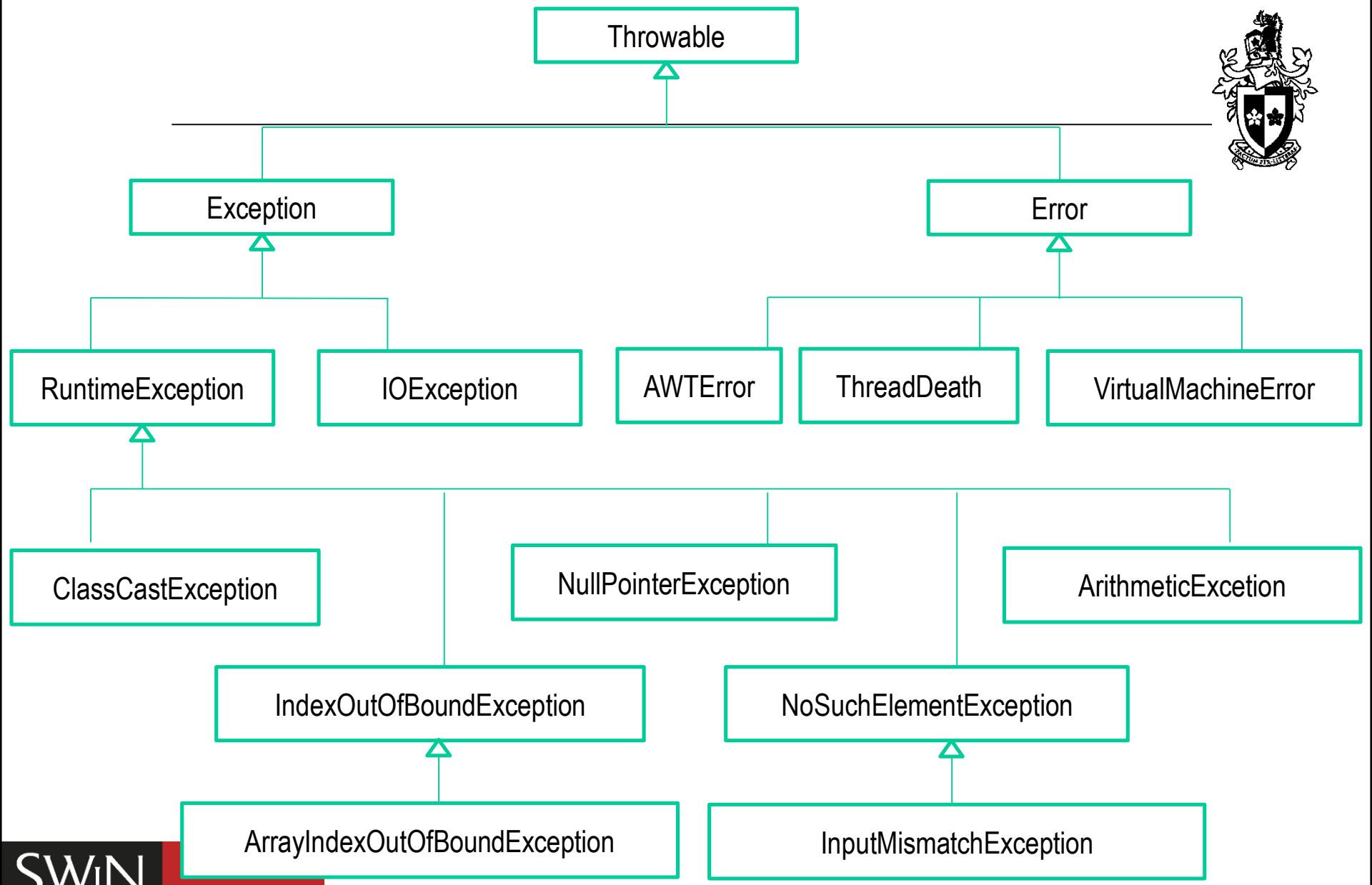
- An exception is an abnormal event that occurs during the execution of a program that disrupts the normal flow of instructions.
- Throwing an exception:
  - When an abnormal event occurs the code creates an exception object and hands it off to the runtime system.
- The exception object:
  - The exception object contains information about the exception, including its type and the state of the program when the error occurred.

Exception: Definition

- Catching an exception:
  - It happens when the exception object bubbles up through the call stack until an appropriate exception handler is found. The handler catches the exception.

Errors: Definition

- When a dynamic linking failure or some other “hard” failure in the virtual machine occurs, the virtual machine throws an Error.



A portion of class Throwable inheritance hierarchy

Throwing an exception

- `Throwable` serves as the base class for an entire family of exception classes, declared in `java.lang` package, that your program can instantiate and throw.
- To throw an exception, you must simply,
  - Construct an exception object
  - Then, use the keyword throw and throw the object

```
throw new SomeThrowableExceptionType();
```

- You can't throw just any object as an exception,however—only those objects whose classes descend from Throwable.

Throwing an exception

```
/**
 * Look up a name or phone number and return the
 * corresponding contact det
 * @param key The name or nu
 * @return The details corresp                    key,
 *         or null if there ar  none matching.
 * @throws NullPointerException if the key is null.
 */
public ContactDetails getDetails(String key)
{
    if(key == null) {
        throw new NullPointerException(
                                "null key in getDetails");
    }
    return book.get(key);
}
```

JavaDoc
`@throws ExceptionType reason`

This is just one type of exception

Its an Object

Normal flow-of-control stops now

The throws clause

- Throws keyword is used when a method contains code that may throw an exception. Then the **throws** clause can be used to declare it.
- It is also used if a method does not handle a checked exception. Then the method must throw it using the **throws** keyword.
- The throws keyword appears at the end of a method header.

```
public void saveToFile(String destinationFile) throws IOException
```

The throws clause

- A method can throw more than 1 exception:

```
import java.io.RemoteException;
public class className
{
   public void withdraw(double amount) throws
                  RemoteException, NullPointerException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

The try catch block

- To catch an exception in java ,we have to write a *try* block with one or more *catch* clauses.
- Each clause specifies one exception type that it is prepared to handle.
- The try block places a fence around a bit of code that is under the watchful eye of the associated catchers.
- If a code delimited by a try block throws an exception
  - The associated catch blocks of that try block will be examined by the JVM
  - If the JVM finds a catch clause that is prepared to handle the thrown exception, the program continues execution starting with the first statement of that catch clause.

The try catch block

- Format

```
try
{
        //Java Statements that might create an exception
}
catch (ExceptionType variablename)
{
        //Java Statements to handle exception
}
```

The try catch block

- Clients catching an exception **must protect** the call with a try statement:

Code that generates an exception must be in enclosed within a try block

Code that might create an exception

```
try
{
    //Protect one or more statements here.
}
catch(Exception e)
{
    //Report and recover from the exception
here.
}
...
```

Catch the exception here

Write recovery code here

After recovery continue normal flow-of-control here

SWINBURNE UNIVERSITY OF TECHNOLOGY

The try catch block

1. Exception thrown by saveToFile method

```
try {
    addressbook.saveToFile(filename);
    tryAgain = false;
}
catch(IOException e) {
    System.out.println("Unable to save to " + filename);
    tryAgain = true;
}
```

2. Control transfers here

Example: Catching an exception

- Observe the following code…

```
public class ExcepTest{

   public static void main(String args[]){
        int a[] = new int[2];
        System.out.println("Access element three :" + a[2]);
   }
}
```

The above is an array declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

Example: Catching an exception

- Solution:

```
public class ExcepTest{

   public static void main(String args[]){
      try{
         int a[] = new int[2];
         System.out.println("Access element three :" + a[2]);
      }
      catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

Catching ArrayIndexOutOfBoundsException

Multiple catch blocks

- A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
    //Protected code
}
catch(ExceptionType1 e1)
{
    //Catch block 1
}
catch(ExceptionType2 e2)
{
    //Catch block 2
}
catch(ExceptionType3 e3)
{
    //Catch block 3
}
```

Example: Catching multiple exceptions

```
try
{
     ...
     ref.process();
     ...
}
catch(EOFException e) {
     // Take action on an end-of-file exception.
     ...
}
catch(FileNotFoundException e) {
     // Take action on a file-not-found exception.
     ...
}
```

You can catch many different exception types (EOFException is a subclass of IOException)

Only the first matching Exception is used

Two types of exceptions

- Checked exceptions
  - Checked exceptions are due to the external circumstances that the programmer cannot prevent
  - The compiler checks that your program handles these exceptions
  - All IOExceptions are checked exceptions
- Unchecked exceptions
  - The unchecked exceptions are the programmer fault and programmer can prevent them
  - They include NullPointerException, IllegalArgumentException, Arithmetic Exceptions etc.
  - for example, to avoid NullPointerException you should check if the references for null

Checked Exception Example – Text input from file

```
try {
    BufferedReader reader =
        new BufferedReader(new FileReader("filename"));
    String line = reader.readLine();
    while(line != null) {
        do something with line
        line = reader.readLine();
    }
    reader.close();
}
catch(FileNotFoundException e) {
    the specified file could not be found
}
catch(IOException e) {
    something went wrong with reading or closing
}
```

Unchecked Exception Example

```
/**
 * Look up a name or phone number and return the
 * corresponding contact details.
 * @param key The name or number to be looked up.
 * @return The details corresponding to the key,
 *         or null if there are none matching.
 * @throws IllegalArgumentException if the key is null or empty
*/
public ContactDetails getDetails(String key)
{
    if(key == null) {
        throw new IllegalArgumentException("null key in getDetails");
    }
    if(key.trim().length() == 0) {
        throw new IllegalArgumentException("Empty key passed to
                                                getDetails");
    return book.get(key);
}
```

The finally clause

- The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.
- Using a finally block allows you to run any cleanup (housekeeping) type of statements that you want to execute, no matter what happens in the protected code.

```
try
{
    //Protect one or more statements here.
}
catch(Exception e)
{
    //Report and recover from the exception here.
}
finally {
    //Perform any actions here whether or not an exception is thrown.
}
```

Handle or Declare Rule

- If some code within a method throws a checked exception, then the method must either handle the exception(using *try-catch* block) or it must specify the exception using *throws* keyword. Else, the code will not compile.

Handle or Declare Rule

- A method catches an exception using a combination of the **try** and **catch** keywords.

```
// File Name : ExcepTest.java
import java.io.*;

public class ExcepTest {
   public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

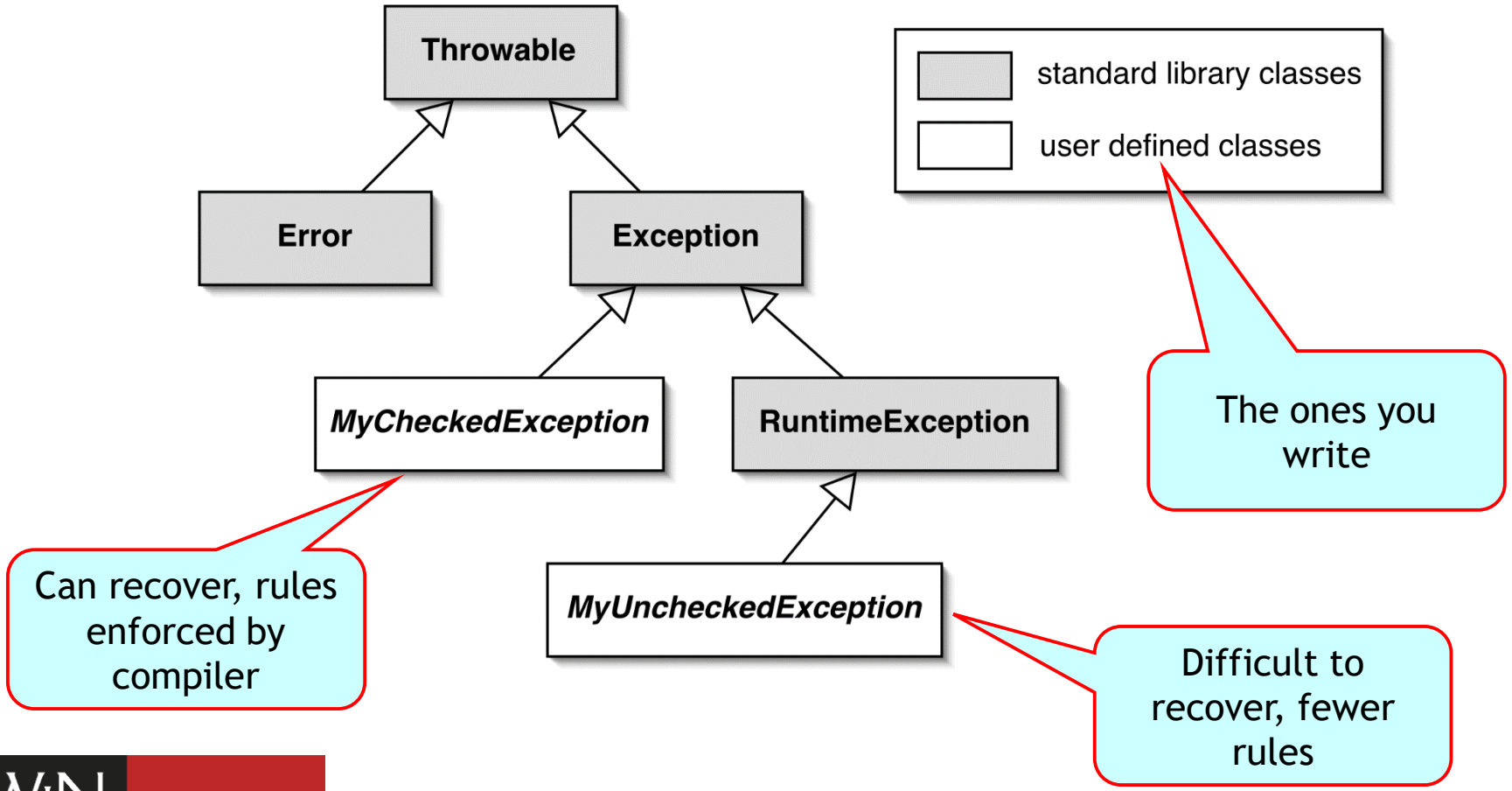
Handle or Declare Rule

- If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

```
import java.io.*;
public class className {

    public void deposit(double amount) throws RemoteException {
        // Method implementation
        throw new RemoteException();
    }
    // Remainder of class definition
}
```

You can create your own Exception class



Defining your own Exception

- Keep the following points in mind when writing your own exception classes:
  - All exceptions must be a child of Throwable.
  - If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.

Defining your Own Exception

```
public class NoMatchingDetailsException extends Exception
{
    private String key;

    public NoMatchingDetailsException(String key)
    {
        this.key = key;
    }

    public String getKey()
    {
        return key;
    }

    public String toString()
    {
        return "No details matching " + key +
               " were found.";
    }
}
```

Define new types to give better diagnostic information. Include reporting and/or recovery information.

Defining your Own Exception

```
/**
* Remove the entry with the given key from the address
book.
* The key should be one that is currently in use.
* @param key One of the keys of the entry to be removed.
* @return true if the entry was successfully removed
* @throws NoMatchingDetailsException if no records found
* for the key
*/
public boolean removeDetails(String key)
{
      if(!keyInUse(key)) {
            throw new NoMatchingDetailsException(key);
      }

      ContactDetails details = book.get(key);
      book.remove(details.getName());
      book.remove(details.getPhone());
      numberOfEntries--;
      return true;
}
```

Exception Throwing

Example:

```
public class Account
{
        public void withdraw(double amount) throws BankException
         {
            if (balance - amount < overdraftLimit)
                    throw new BankException("overdrawn");
            balance -= amount;
         }
```

Not executed if throw is done.

Declaring Exceptions

Example:

```
public class BankException extends Exception
{
        public BankException (String s)
                { super(s); }
}
```

Use a defensive programming approach

- Anticipating that things could go wrong
- Managing what goes wrong
- Keeping the user informed
- Shutting down gracefully, if needed

SWINBURNE UNIVERSITY OF TECHNOLOGY

What defensive measures I could take ?

Check Parameter values

- Parameters represent a major 'vulnerability' for a business object.
  - Constructor parameters initialize state.
  - Method parameters often contribute to behavior.
- Parameter checking is one defensive measure.

```
public void removeDetails(String key)
{
        ContactDetails details = book.getDetails(key);
        book.remove(details.getName());
        book.remove(details.getPhone());
        numberOfEntries--;
}
```

We should check the parameter passed to the method, before proceeding

Check parameter values

```
public ContactDetails getDetails(String key)
{
    if(key == null) {
        throw new IllegalArgumentException(
                            "null key in getDetails");
    }
    if(key.trim().length() == 0) {
        throw new IllegalArgumentException(
                          "Empty key passed to getDetails");
    }
    return book.get(key);
}
```

Here we are checking if the parameter is Null

Here we are checking if the parameter is a empty string

We can throw different exceptions after checking different pre conditions

SWINBURNE UNIVERSITY OF TECHNOLOGY

Preventing object creation if the state is invalid

We can throw an exception if the parameters passed to the constructor prevent objects from being created with a valid initial state.

```
public ContactDetails(String name, String phone, String address)
{
   // Use blank strings if any of the parameters is null.
    if(name == null) {
       name = "";
    }
    if(phone == null) {
       phone = "";
    }
    if(address == null) {
       address = "";
    }
    this.name = name.trim();
    this.phone = phone.trim();
    this.address = address.trim();

    if(this.name.length() == 0 && this.phone.length() == 0) {
        throw new IllegalStateException(
                        "Either the name or phone must not be blank.");
    }
}
```

If the parameters passed to the constructor is not valid, then throw an Exception…

Exception Danger areas

- File Writing / Reading
- BufferedReader and PrintWriter
- Reference variables that have not been assigned an object
- Incorrect casting, one object type to another
- Text to other types (int, double)
- An inappropriate client call to a server object.
- And many more…

SWINBURNE UNIVERSITY OF TECHNOLOGY

How can I recover from an exception?

Recovering from Errors/ Exceptions

- Client code should take note of error notifications.
  - Check return values.
  - Don't 'ignore' exceptions.
- Include code to attempt recovery.
  - Will often require a loop.

Attempting recovery

```
// Try to save the address book.
final int MAX_ATTEMPTS = 3;
boolean successful = false;
int attempts = 0;
String filename = "AddressBook.txt";
do {
    try {
        addressbook.saveToFile(filename);
        successful = true;
    }
    catch(IOException e) {
        System.out.println("Unable to save to " + filename);
        attempts++;
        if(attempts < MAX_ATTEMPTS) {
            filename = give an alternative file name;
        }
    }
} while(!successful && attempts < MAX_ATTEMPTS);
if(!successful) {
    Report the problem and give up;
}
```

Another Example

```
do {
    try {
        System.out.print("Enter coordinates of vertices of triangle…");
        x1 = Keyboard.readInt("x1");
        . . .
        y3 = . . . . . . . . . . . . . .   ("y3");
        calculate();
        ok = true;

    }

    catch (ArithmeticException e) {
        System.out.println("***problem with that triangle.  Try again.");
    }

} while (!ok);
```

Reading

- https://docs.oracle.com/javase/tutorial/essential/exceptions/
- Barnes & Kolling Chapter 12

SWINBURNE UNIVERSITY OF TECHNOLOGY