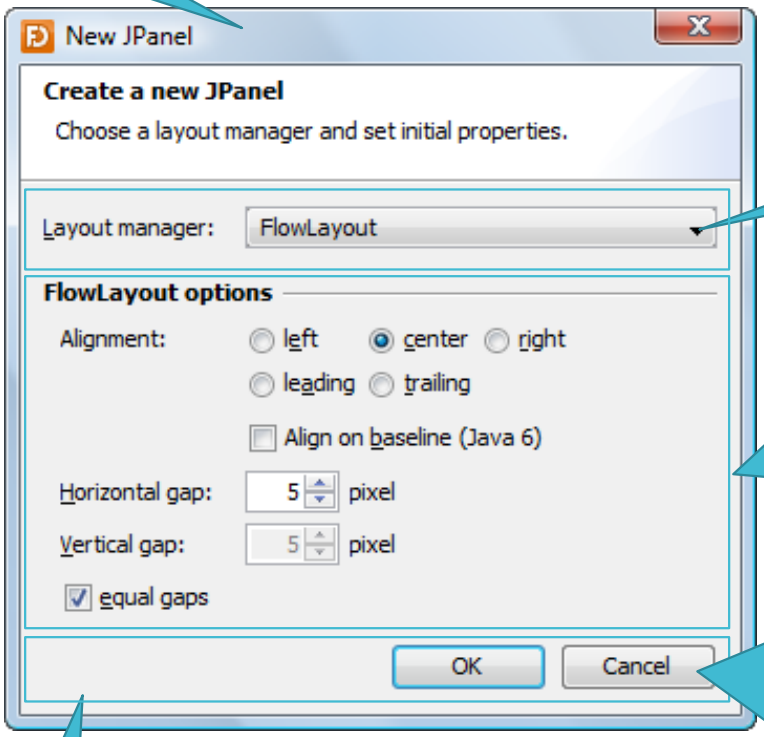
Create GUI & Handle Event

Lab Objectives

Create GUI using simple components & handle events

Java GUI

Window



Components have properties

Layout

- Flow Layout
- Border Layout
- Grid Layout
- …

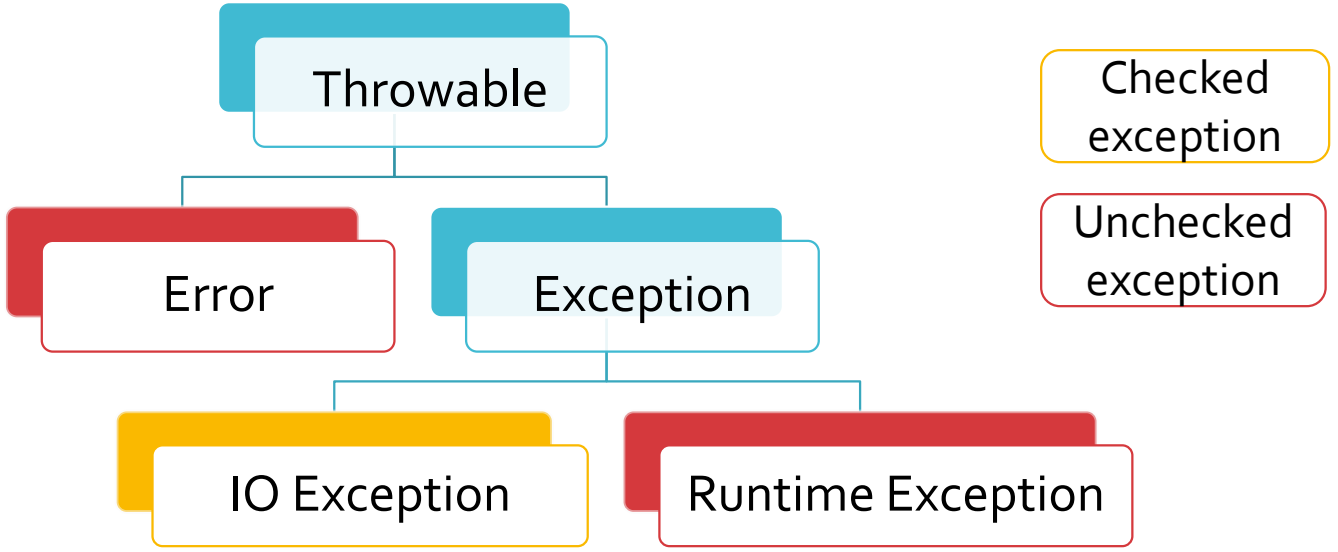
Listener:

- Action Listener
- Change Listener
- Item Listener
- Component Listener
- Focus Listener
- Key Listener
- Mouse Listener
- …

Panel

Java Exceptions

Exception Hierarchy



Common built-in exception:

- **ArithmeticException** (e.g., divide by zero)
- **ClassCastException** (e.g., attempt to cast a String Object to Integer)
- **IndexOutOfBoundsException**
- **NullPointerException**
- **FileNotFoundException** (e.g., attempt to open a non-existent file for reading)

Catching Exceptions

- A **try/catch** block is placed around the code that might generate an exception
- A **finally** block of code that follows a **try** block, always executes, whether or not an exception has occurred.
- Example:

```
try {
   //Protected code
}catch(ExceptionType1 e1) {
   //Catch block
}catch(ExceptionType2 e2) {
   //Catch block
}catch(ExceptionType3 e3) {
   //Catch block
}finally {
   //The finally block always executes.
}
```

throws/throw Keywords

- If a method does not handle a *checked exception*, the method must declare it using the **throws** keyword.
- You can **throw** an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword
- Example:

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws  RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

Declaring you 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, extend the **Exception** class.
  - If you want to write a runtime exception, extend the **RuntimeException** class.

```
public class MyException extends Exception
{
   private double field1;
   public MyException (double input)
   {
      this.field1 = input;
   }
   public double getField1 ()
   {
      return field1;
   }
}
```

When to throw an exception

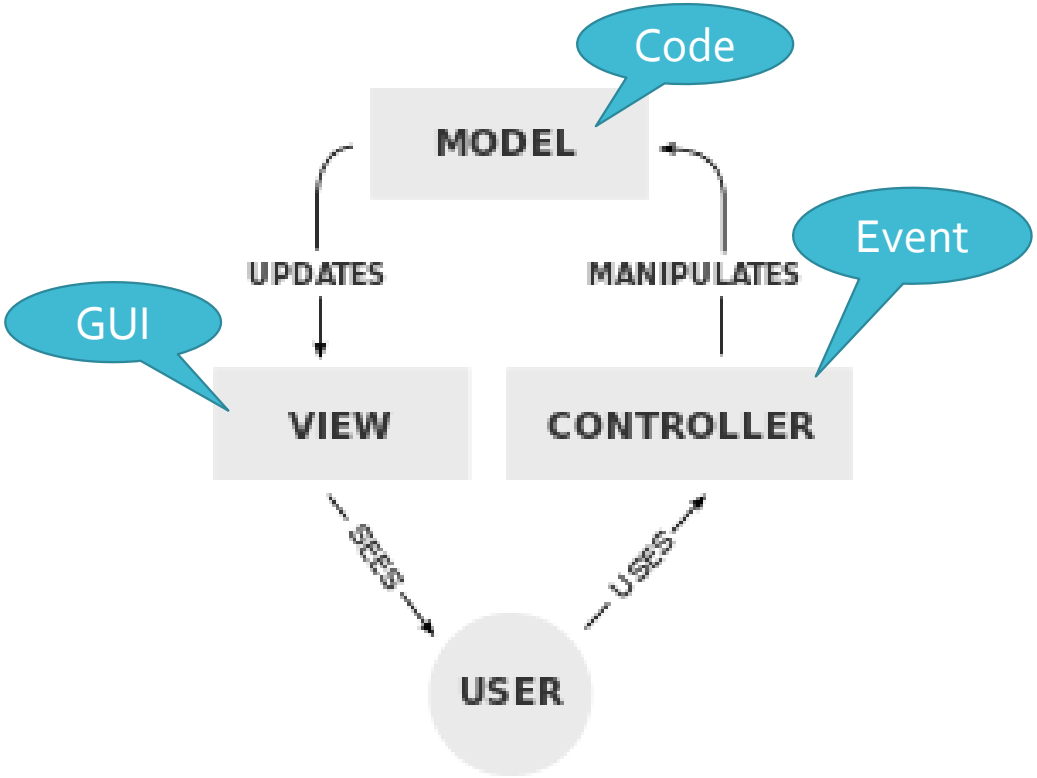
- "An exception is thrown when a fundamental assumption of the current code block is found to be false."
- "The other side of this equation is: if you find your functions throwing exceptions frequently, then you probably need to refine their assumptions."

http://stackoverflow.com/questions/77127/when-to-throw-an-exception

GUI SAMPLE PROGRAM

- Exercise 1: Hello Word
- Exercise 2: Converter

MVC model



- Model-View-Controller design pattern
- Isolate data model from user interface from application logic
  - **Model**: data model classes
  - **View**: user interface
    (i.e. GUI, console, etc.)
  - **Controller**: interacts with View, manipulates Model

Exercise: Converter

- **Problem:**
  - Create a GUI program to convert Fahrenheit to Celsius $(32°\mathbf{F} - 32) \times 5/9 = 0°\mathbf{C}$
  - **Model (Data):**
  - We don't have to store any data so no need
- **View (GUI):**
  - 1 label
  - 1 input text
  - 1 button
  - 1 display text
- **Controller (User interactive)**
  - Click the button

To control more about the window and can be reusable, we create our own object extending JFrame

input Temp not less than - 459.67 degree