# Throwing Exceptions

You can create and throw your own exceptions in code.

## What to do when something is wrong?

The insert( Coin ) method of Purse requires that a coin have a positive value and a currency.

What if coin has 0 value or no currency?

1. Simply ignore it
2. Return a special value to indicate failure.
3. Print a message on console.
4. Throw an exception

Let's compare the choices...

## Insert an illegal coin?

1. Simply ignore it

The worst solution -- the code has no idea that something is wrong.

2. Return a special value to indicate failure.

OK, but then the caller <u>must always check</u> the return value. Sometimes this is reasonable. Usually it makes the code more complex & harder to read.

```
boolean insertOk = purse.insert( coin );
if (! insertOk ) {
   System.out.printf("Sorry, can't insert %s",
   coin.toString()));
   // why not?  What's wrong?
```

## Print a Message?

3. Print a message on the console

Bad idea

a) calling method doesn't see the problem

b) there may not be a console (web app, mobile app)

c) doesn't help solve the problem

```
Design Principle:
    "throw exceptions, don't print them"
```

## Throw Exception

### 4. Throw an exception

For unusual conditions and errors, the cleanest solution is often to throw an exception.

```
/** Add two money objects.
 *  @param other another money with same
 *      currency to add to this money object.
 */
public Money add(Money other) {
   if (! this.currency.equalsIgnoreCase(
              other.getCurrency()) {
   throw new IllegalAgumentException(
     "Cannot add money with different currency");
   }
```

## Commonly Used Exception

IllegalArgumentException("message") - parameter to a method or constructor violates requirements

UnsupportedOperationException - the requested operation is not supported.  This is thrown by optional methods like Iterator remove().

RuntimeException("message") - a general catch-all for conditions that don't have a specific exception.

## How to Create Exception?

Exceptions are ordinary Java classes that extend Exception or RuntimeException.
You can create instances using the usual Java syntax.
Exceptions usually have 4 constructors:

```
Exception(String message)
```
- provide a description message of cause

```
Exception(Throwable other)
```
- wrap another exception

```
Exception(String message, Throwable cause)
```
- wrap another exception and provide an explanatory message.

```
Exception()
```
- OK if exception class makes the cause clear

## Stop Bad Money - throw exception

In Coin Purse, we can stop bad money by having the top-level class (Money or AbstractValuable) constructor validate the parameters & throw exception.

```
/**
 * Money with a value and currency.
 * @param value the value of money, must be pos.
 * @throws IllegalArgumentException if invalid
 */
public Money(double value, String currency) {
    if (value <= 0.0)
       throw new IllegalArgumentException(
         "Value of money must be positive");
```

## "Wrap" an Exception

Catch an exception and rethrow it inside another exception object.

```
/** Read all data from an InputStream */
public String readAll(InputStream in) {
    try {
        // read the entire input
        // InputStream may throw IOException
    } catch (IOException ex) {
       throw new RuntimeException(
            "Exception reading input", ex);
    }
}
```

## Why "Wrap" an Exception?

You can catch an exception and "wrap" it in another exception, then throw it.

Why do this?

1. Convert a Checked Exception to Unchecked

   - so caller is not required to use "try - catch"

2. Provide a more meaningful exception type

## What Exception Type to Use?

See if you can find a suitable exception in the Java API.

Generally prefer to use an unchecked exception, namely, RuntimeException and its subclasses.

If necessary, create your own subclass of RuntimeException.

Example:  a Stack class might define a StackFullException

## Useful: IllegalArgumentException

```
public class Money implements Valuable {
  /**
   * Instantiate Money with a value and currency.
   * @param value of the money, may not be neg.
   * @throws IllegalArgumentException
   *         if value is negative
   */
  public Money(double value, String currency) {
     if (value < 0.0)
       throw new IllegalArgumentException(
          "Value may not be negative");
     this.value = value;
     ...
```

## Rethrowing the *Same* Exception

A function can catch an exception and throw it again.

Sometimes used for logging.

```
try {
    sub();   // sub() throws exception
}
catch ( RuntimeException e ) {
    // log the problem
    Logger.getLogger().warning(
        "sub() threw exception: "+ e );
    // throw it again!
    throw e;
}
```