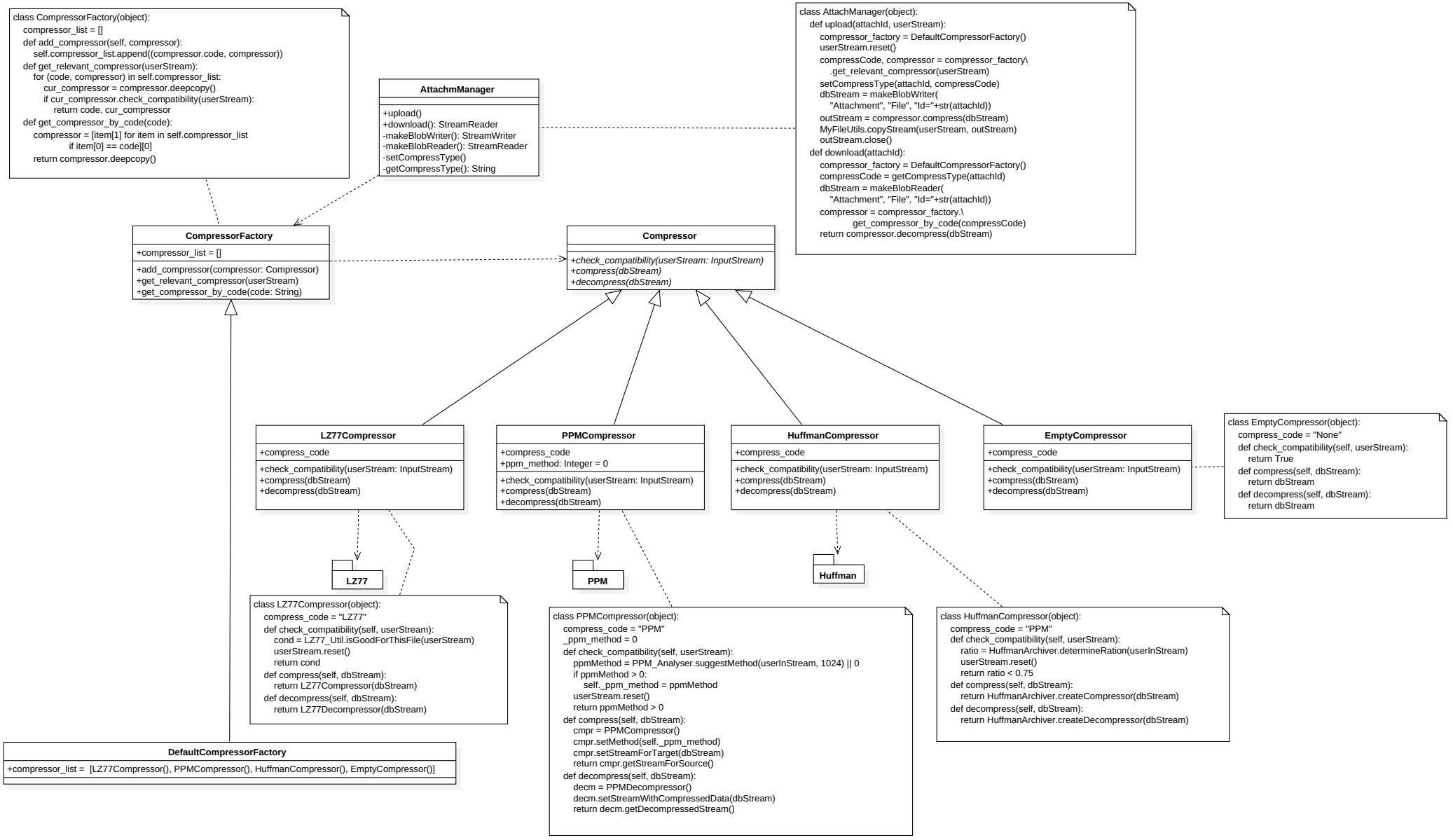
Model::Main


class CompressorFactory(object):
compressor_list = []
def add_compressor(self, compressor):
self.compressor_list.append((compressor.code, compressor))
def get_relevant_compressor(userStream):
for (code, compressor) in self.compressor_list:
cur_compressor = compressor.deepcopy()
if cur_compressor.check_compatibility(userStream):
return code, cur_compressor
def get_compressor_by_code(code):
compressor = [item[1] for item in self.compressor_list
if item[0] == code][0]
return compressor.deepcopy()
AttachmManager
+upload()
+download(): StreamReader
-makeBlobWriter(): StreamWriter
-makeBlobReader(): StreamReader
-setCompressType()
-getCompressType(): String
class AttachManager(object):
def upload(attachId, userStream):
compressor_factory = DefaultCompressorFactory()
userStream.reset()
compressCode, compressor = compressor_factory\
.get_relevant_compressor(userStream)
setCompressType(attachId, compressCode)
dbStream = makeBlobWriter(
"Attachment", "File", "Id="+str(attachId))
outStream = compressor.compress(dbStream)
MyFileUtils.copyStream(userStream, outStream)
outStream.close()
def download(attachId):
compressor_factory = DefaultCompressorFactory()
compressCode = getCompressType(attachId)
dbStream = makeBlobReader(
"Attachment", "File", "Id="+str(attachId))
compressor = compressor_factory.\
get_compressor_by_code(compressCode)
return compressor.decompress(dbStream)
CompressorFactory
+compressor_list = []
+add_compressor(compressor: Compressor)
+get_relevant_compressor(userStream)
+get_compressor_by_code(code: String)
Compressor
+check_compatibility(userStream: InputStream)
+compress(dbStream)
+decompress(dbStream)
LZ77Compressor
+compress_code
+check_compatibility(userStream: InputStream)
+compress(dbStream)
+decompress(dbStream)
PPMCompressor
+compress_code
+ppm_method: Integer = 0
+check_compatibility(userStream: InputStream)
+compress(dbStream)
+decompress(dbStream)
HuffmanCompressor
+compress_code
+check_compatibility(userStream: InputStream)
+compress(dbStream)
+decompress(dbStream)
EmptyCompressor
+compress_code
+check_compatibility(userStream: InputStream)
+compress(dbStream)
+decompress(dbStream)
class EmptyCompressor(object):
compress_code = "None"
def check_compatibility(self, userStream):
return True
def compress(self, dbStream):
return dbStream
def decompress(self, dbStream):
return dbStream
LZ77
Huffman
PPM
class LZ77Compressor(object):
compress_code = "LZ77"
def check_compatibility(self, userStream):
cond = LZ77_Util.isGoodForThisFile(userStream)
userStream.reset()
return cond
def compress(self, dbStream):
return LZ77Compressor(dbStream)
def decompress(self, dbStream):
return LZ77Decompressor(dbStream)
class PPMCompressor(object):
compress_code = "PPM"
_ppm_method = 0
def check_compatibility(self, userStream):
ppmMethod = PPM_Analyser.suggestMethod(userInStream, 1024) || 0
if ppmMethod > 0:
self._ppm_method = ppmMethod
userStream.reset()
return ppmMethod > 0
def compress(self, dbStream):
cmpr = PPMCompressor()
cmpr.setMethod(self._ppm_method)
cmpr.setStreamForTarget(dbStream)
return cmpr.getStreamForSource()
def decompress(self, dbStream):
decm = PPMDecompressor()
decm.setStreamWithCompressedData(dbStream)
return decm.getDecompressedStream()
class HuffmanCompressor(object):
compress_code = "PPM"
def check_compatibility(self, userStream):
ratio = HuffmanArchiver.determineRation(userInStream)
userStream.reset()
return ratio < 0.75
def compress(self, dbStream):
return HuffmanArchiver.createCompressor(dbStream)
def decompress(self, dbStream):
return HuffmanArchiver.createDecompressor(dbStream)
DefaultCompressorFactory
+compressor_list = [LZ77Compressor(), PPMCompressor(), HuffmanCompressor(), EmptyCompressor()]