# COSC 2P05 — Assignment 1 — Save Our City

*Save our city*
*Keep our souls, Lord*
*Through the rapture of this world*

For this assignment, we'll be looking at different ways of abstracting data types, and using polymorphism.
More specifically, you'll be writing a program to simulate a zombie apocalypse, wherein a rampaging horde of the `Undead` will be shambling continuously towards a ragtag band of `Survivor`s making a last stand.
The user will be able to select things like how many zombies there are (and how many may approach simultaneously), and who will fight them.

To get a better understanding of the mechanics (if not ideal design) of different abstract types, you'll be using an *abstract class* to define a `Survivor`, and an *interface* to define an `Undead`. Because the point is to understand the mechanisms, rather than to be a design task, you're provided with samples of both, as well as a temporary, **incomplete** skeleton of a main class. Additionally, you've been provided with a helper class (`Horde`) to assist with all your reanimated needs.

Zombies can only attack after closing the distance (once they reach a distance of 0, after having started 4 spaces away), and may only attack whomever is in front. Only the hero at the front may act, though each has different actions/targets.

The zombies come in the following varieties:

- `CHUD`s:
  - Slow (can only move 1 space at a time)
  - Weak (can only inflict 1HP damage per attack)
  - *Always* hungry (can attack each turn once within range)
- `Deadite`s:
  - Slow (can only move 1 space at a time)
  - Weak (can only inflict 1HP damage per attack)
  - Gets full easily (can only `chomp` Survivors every other turn)
- `Trioxin` zombies:
  - Fast (can move 2 spaces at a time)
  - Strong (can inflict 2HP per attack)
  - Gets full easily (can only `chomp` every other turn)
  - Has a custom `speak` string: *Send more paramedics*

The heroes may be selected from:

- `Plinker`:
  - Fires three shots; each of which does 1 'limb' of damage
  - Targets are chosen randomly (degree of 'randomness' doesn't really matter)
- `Sniper`:
  - Picks a target (e.g. randomly); eliminates that target (full 4 limbs of damage)
- `Flamer`:
  - Can only harm zombies that are distance of zero, but insta-kills all such zombies
- `Medic`:
  - Does no damage, but completely heals the party

Refer to the provided files for additional insight.
Note that each `Survivor` must also be instantiated with a name. People often have names.

## Requirements

To consolidate:

- You need to either replace `Apocalypse.java`, or *substantially* supplement it
  - The user needs to be able to decide on the makeup of the human side, how many zombies to create altogether, and how many may be 'in play' at a time
  - The user should be able to choose a number of 'waves' (iterations of humans acting, zombies advancing, zombies acting) to run at a time
  - The user should be able to decide when to start over, or quit
  - Basically, the marker needs to be able to test an actual, complete program
- You *may* wish to tweak `Horde`
  - It's written to confirm that basic functionality works; not to be 'good'
- You need to provide all of the implementation classes promised

## Tips and Suggestions

Of course, the most important thing is to work out what you're doing *long* before you start coding.
However, in addition to that:

- This isn't a data structures course. You're welcome to use Java's built-in `LinkedList`, etc.
- Of course, you need basic conventions:
  - Put your **name, username and student number** at the top **of each file** in commenting
  - Document each class and method, in JavaDoc convention
- Everything goes into an `(apocalypse)` package
  - Actually, it's up to you whether or not you want your Main class/program to be in the same package or a different one, but everything *else* does need to be in there
- Take a look at the marking scheme below. What do you see?
  - You aren't graded on how much time you put into the assignment, nor on whether or not the output vaguely resembles what one might expect. You're being told to create — and use — a well-defined specification. Cobbling together bits and bobs isn't sufficient
  - Also, some things are explicit, and some aren't; don't assume there's "one way" to solve a problem
- Corollary to the previous point: if you have a single `print` statement **anywhere** other than within your Main class, don't be surprised when you don't receive a passing grade

## Grading

This assignment is out of **10**. Half-points are possible.

Points are earned as follows:

**The `Apocalypse` (main program) /3**

- Note that you need to include a menu for this
- Most actions have `String` return values. You need to work these in somehow as feedback to the user

**The `Flamer`, `Sniper`, `Medic`, and `Plinker`: /2**

- Collectively out of 2, because solving one is most of the work of solving the rest

**The `Deadite`, `CHUD`, and `Trioxin`: /2**

- Yes, out of 2; I didn't feel like 1.5
- Fill in the requirements, including logic for handling hunger (remembering that `CHUD`s are special!)

**The `Horde`:**		**/1**

- Yes, you can get this for just submitting what you were given

**Additional:**		**/2**

- There are several things that couldn't fit into one of the above, including style, coding and commenting conventions, etc.
- This is basically a catch-all for being thorough and reasonable in your designs

**Warning: additional points can also be deducted!**
There are certain expectations that you aren't rewarded for following. (e.g. you're expected to realize that zombies can't have 'negative limbs', humans can't be super-killed after already being dead, zombies can't just run straight past their prey and fall off into negative-distance-ness, etc.)

However, deductions outside of the above will only be made for things that are **objectively wrong**.

Also: **no BasicIO.jar stuff**. The marker won't have it, so you can't use it. Standard Java only.

## Submission

You'll be submitting through Sakai.
Bundle everything up into a single **`.zip`** file, and submit that.
Note: `.zip`.
**Not `.rar`**.
Not .7z.
This is not 1993.

Ideally, your submission (including project folder) will be simple. If not, the burden is on you to ensure that the marker can compile and run it. It *must* be able to be run on either a lab computer or sandcastle.