

Learning Outcomes

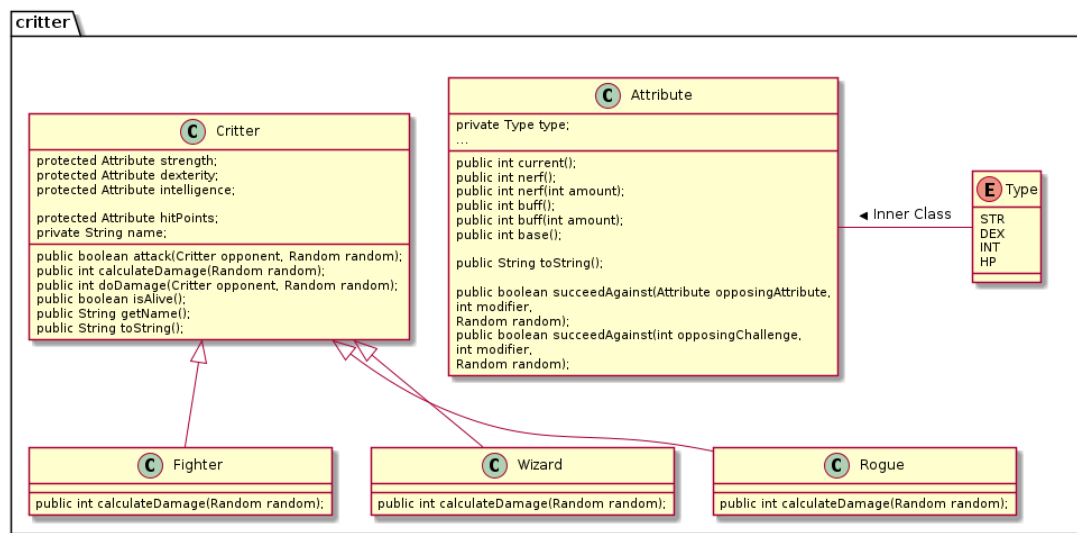
Upon completing this assignment, students should be able to

- **Implement** a class *hierarchy* where subclasses use *extension points* to modify the behavior of the base class.
- **Use** a *factory method* to generate a random instance of one of the subclasses.
- **Use** Random to generate random integer values.
- Write a function that runs a simple “combat” between two randomly created critters until one of them has zero or fewer hit points.

Introduction

This program is the first of a series of programs that will implement a very simple RPG (role playing game) combat system. For this installment you will create a hierarchy of classes that implement different types of Critters, each with four Attribute, generate two *random* Critter, and have them fight until one is dispatched.

The classes to be implemented in the critter package are shown below. The Type class should be a local class within Attribute; the drawing program has a limitation so it is drawn outside.



None of the classes show their *constructors*; the primary constructor for each class is one that takes all the values necessary to initialize the fields of the object (and its ancestor objects). You may, for convenience, define other constructors that provide default values to all of the fields.

Notice that the above says all **necessary** values. Some values might have obvious default values (think about the buff and nerf values inside an Attribute; no matter how they are actually stored, the starting value seems like it should be zero for both values).

The methods for Attribute and Critter are described below:

Attribute

base() return the *base* value of this attribute without any advantages or disadvantages applied.

current() return the current value of this attribute, including all advantages and all disadvantages.

nerf() return the total *magnitude* of this attribute's **disadvantages**. Can never go below zero.

nerf(int amount) change total disadvantages by the given amount. If amount is *positive*, the attribute will be *more* disadvantaged and if it is *negative*, the attribute will be *less* disadvantaged. Cannot adjust value below zero. Return what **nerf()** would return just after the adjustment.

buff() return the total *magnitude* of this attribute's **advantages**. Can never go below zero.

buff(int amount) change total advantages by amount, never going below zero. Return post-adjustment **buff()** value.

toString() return a string of the form

NAM: (<a>/-<d>)

where NAM is the name of the attribute, is the base value, <a> is the **buff()** value, and <d> is the **nerf()** value.

succeedAgainst(Attribute opposingAttribute, int modifier, Random random) run an *opposed* test of this attribute (another critter is trying to make the attempt fail). **opposingAttribute** is the **Attribute** of the opponent, the **modifier** is a situational adjustment on this test, and the **random** is used for rolling a d20.

The test succeeds if

`this.current() + d20 + modifier - opposingAttribute.current() > 10`

succeedAgainst(int opposingChallenge, int modifier, Random random) run an *unopposed* test where the **opposingChallenge** is the intrinsic difficulty, **modifier** is a situational modifier, and **random** is for rolling a d20.

The test succeeds if

`this.current() + d20 + modifier - opposingChallenge > 10`

Critter

getName() return the critter's name.

isAlive() does the critter have a *positive* number of hit points?

attack(Critter opponent, Random random) **attempt** to attack (hit) opponent with some sort of test using **random** to roll the dice. This method does **not** apply damage or the like, it just determines whether or not the attack, whatever it was, succeeds. By default, this is a **STR/STR** challenge.

calculateDamage(Random random) return the number of points of attribute damage the critter did on a successful attack. **random** is for rolling dice. For now, each character class has a different damage range: Fighter: 1d8; Rogue: 1d6; Wizard: 1d4+1.

doDamage(Critter opponent, Random random) apply damage to opponent by calling **calculateDamage**. Return the damage point value.

battle.Arena

The **battle.Arena**, the main program for this project, will generate two *random* critters (without doing anything smart about attribute allocation) and then run through combat *rounds* until one of the critters is dead.

You should make a *factory method* in **Arena**,

`Critter critterFactory(String kind, String name, int STR, int DEX, int INT)`

kind is a string naming one of the **Critter** subtypes, the one to construct. The three integers are the initial base values for the three attributes of the new critter. Note that HP is determined by subclass: Fighter = 8, Rogue = 6, Wizard = 4. Using an **if**, call the appropriate constructor and return the result.

```
Critter makeRandomCritter(String name, Random random)
```

uses random to pick one of the critter types (even probability at this point) and generate three starting attribute values by rolling three d6. This means calling for a random number 1-6 three times and summing the results. Then call the factory with the attributes and return its results.

```
Critter fight(Critter combatantA, Critter combatantB, Random random)
```

runs a battle between the two critters given, using random to roll necessary dice. The fight takes place in a series of *rounds*. The overall fight looks something like this:

```
while (both critters are alive) {
    each critter rolls a d20 until they are different:
        assign high number to first
        assign low number to second

    if (first.attack(second, random))
        first.doDamage(second, random);
    if (second.isAlive())
        if (second.attack(first, random))
            second.doDamage(first, random);

    return still living critter (or null)
}
```

Example Output

Note that the names for the *first* Critter and the *second* Critter are hard-wired. The critter attribute values are randomized.

```
$ gradle run -q --console=plain
Let's Get Ready to Rumble!
Wizard: Eric the Tan
Attributes--
STR: 10 (+0/-0)
DEX: 11 (+0/-0)
INT: 13 (+0/-0)
HP: 4 (+0/-0)
```

```
versus
Fighter: Guldorn the Clumsy
Attributes--
STR: 12 (+0/-0)
DEX: 8 (+0/-0)
INT: 11 (+0/-0)
HP: 8 (+0/-0)
```

Guldorn the Clumsy attacks Eric the Tan and HITS for 6 points
Guldorn the Clumsy drops Eric the Tan, winning the battle.

And again:

```
$ gradle run -q --console=plain
Let's Get Ready to Rumble!
Fighter: Eric the Tan
Attributes--
STR: 13 (+0/-0)
DEX: 12 (+0/-0)
INT: 10 (+0/-0)
HP: 8 (+0/-0)
```

versus

```
Wizard: Guldorn the Clumsy
Attributes--
STR: 13 (+0/-0)
DEX: 12 (+0/-0)
INT: 9 (+0/-0)
HP: 4 (+0/-0)
```

```
Guldorn the Clumsy attacks Eric the Tan and misses
Eric the Tan attacks Guldorn the Clumsy and HITS for 3 points
Eric the Tan attacks Guldorn the Clumsy and misses
Guldorn the Clumsy attacks Eric the Tan and HITS for 2 points
Eric the Tan attacks Guldorn the Clumsy and HITS for 4 points
Eric the Tan drops Guldorn the Clumsy, winning the battle.
```

Deliverables

Submission medium: git to Gitea at `cs-devel.potsdam.edu`. `pMakingMenu` is the name of the repo. So, the repo, in the `drbcladd` repo would be: `S24-205-drbcladd/pMakingMenu.git`; the naming is important so that I can have a program download your work.