Assignment #8 – Polymorphism

Due at next Wed 23:59:59

Introduction to Computers II

Tasks

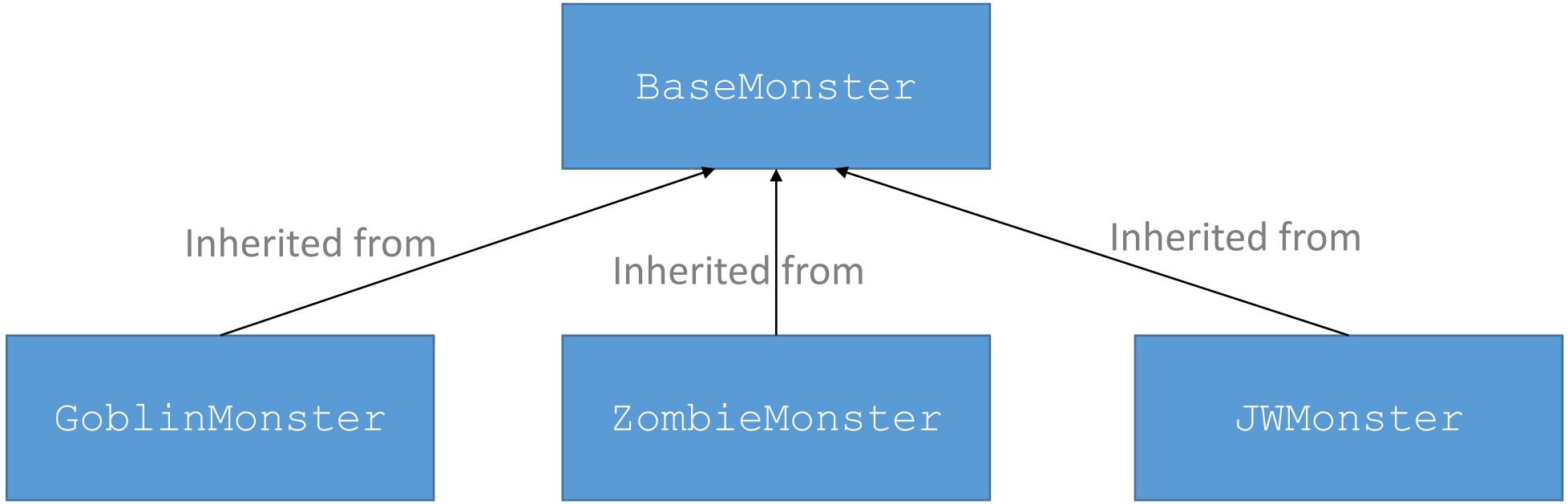
- Modify `NovicePlayer` and its derived classes
- Design `BaseMonster` and its derived classes
  - You can add more derived classes if you'd like to
- Implement serialization on the classes above

Modifying `NovicePlayer` and its derived classes

- Modify the `setLevel()` to a virtual function
- Remove `heal()` in `KnightPlayer` and `pray()` in `MagicianPlayer`
- Implement public virtual function `void specialSkill()`
  - `heal()` in `KnightPlayer`
  - `pray()` in `MagicianPlayer`
  - For `OrcPlayer` and `NovicePlayer`, this function does nothing
- Add a virtual function to `NovicePlayer`
  - `string serialize()`
    - We will describe this later

Implement class `BaseMonster` and its derived classes

- You can add more classes if you'd like to
- Sample inheritance hierarchy:



Data members of `BaseMonster`

- `public const` data members
  - `string name;` // Name of the monster
  - `int attack;` // Attack of the monster
  - `int defense;` // Defense of the monster
  - `int exp;` // Experience earned by players after beating this monster
  - `int money;` // Amount of money dropped after beating this monster
  - `int max_hp;` // The monster's maximum HP
  - `Int max_mp;` // The monster's maximum MP

Why `public const`?

- Because monsters does not have “level”, the attributes of name, attack, defense… are fixed and will not get modified
- So we just set them as constants and open to access from outside

- Think: How can we initialize derived classes of `BaseMonster` with different values of these `public const` members?

Data members of `BaseMonster`

- `private` data members
  - `int hp;` // Current HP of this monster, range: [0, `max_hp`]
  - `int mp;` // Current MP of this monster, range: [0, `max_mp`]

- `static private` data members
  - `int count;` // Number of instances of monster series classes
    // Don't forget to increase/decrease it within proper places

Constructor and Destructor

- `AbstractMonster(string, int, int, int, int, int, int)`
  - `string name`
  - `int attack`
  - `int defense`
  - `int exp`
  - `int money`
  - `int max_hp`
  - `int max_mp`
- `~AbstractMonster()`

Member Functions of `BaseMonster`

- Public member functions
  - `void setHP(int)`
  - `int getHP() const`
  - `void setMP(int)`
  - `int getMP() const`

- Please note that there are limits on the range of `hp` and `mp`, don't forget to check them

Member Functions of `BaseMonster`

- Public static member functions (class methods)
  - `int BaseMonster::getInstanceCount(void)`
    - This method returns the `BaseMonster::count` value.

- **Pure virtual** function
  - `string serialize() = 0;`
    - We will describe this later

Predefined Values

	GoblinMonster	ZombieMonster	JWMonster
name	Goblin	Zombie	JWMaster
attack	60	50	120
defense	40	65	100
exp	12	17	42
money	30	65	175
max_hp	100	150	250
max_mp	50	30	100

- All values can be changed according to your own favor ☺

Serialization

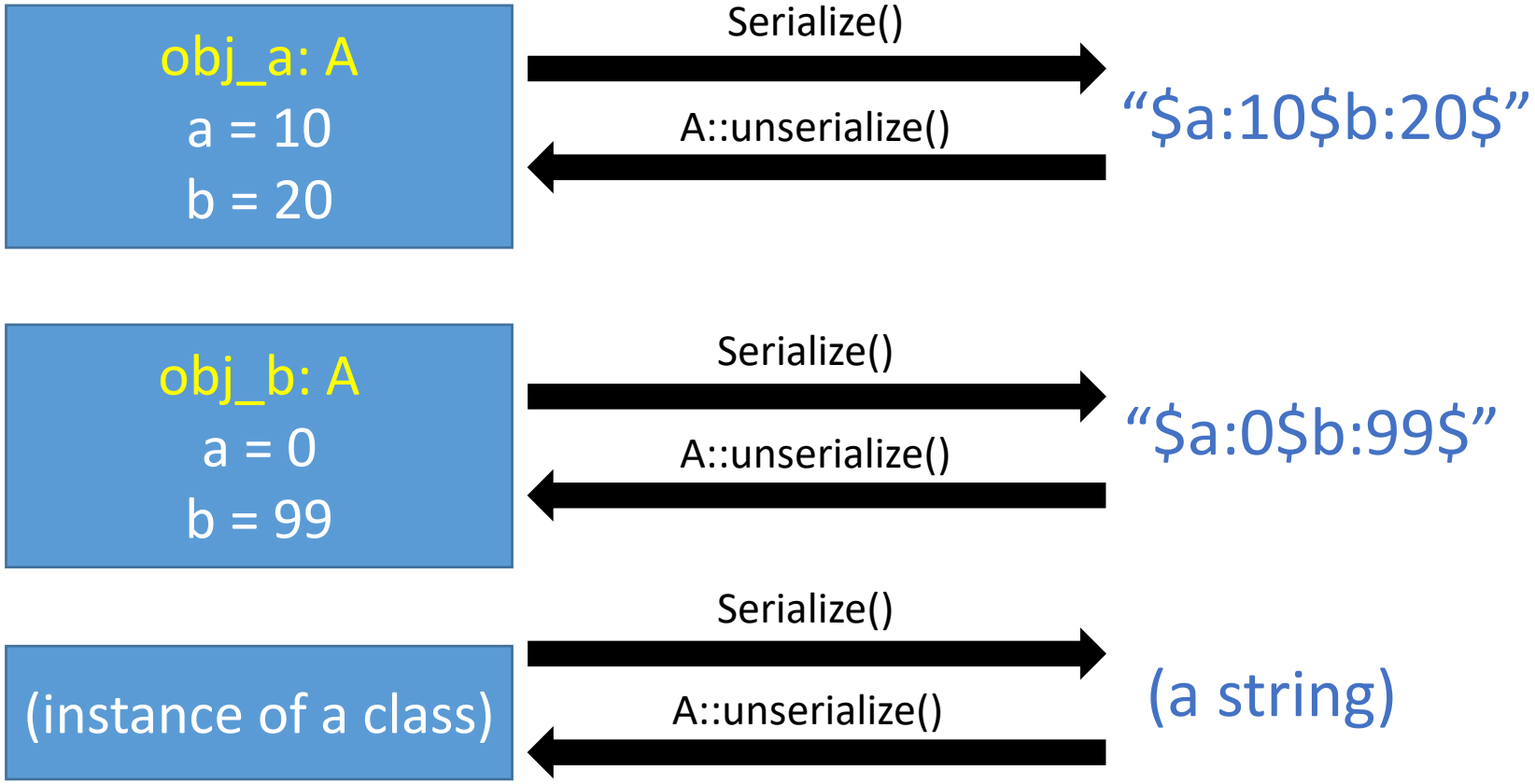
- Serialization is the process of translating data structures or object state into a format that can be stored and reconstructed later in the same or another computer environment
  - E.g., string, XML, JSON…

- We can further implement save and load features using this function in the future

Serialization: Example

```
class A {
private:
    int a;
    int b;
    …
};

A obj_a;
A obj_b;
...
```



Serialization: Tasks

- Please implement `string serialize()` method in derived classes of `NovicePlayer` and `BaseMonster`
  - The format of the returned string is not specified

- Please note that `NovicePlayer` is a class that able to instantiate, so you also need to implement `serialize()` for `NovicePlayer`

Serialization: Tasks

- Implement `static NovicePlayer* unserialize(string)` method in derived classes of `NovicePlayer`
- Implement `static BaseMonster* unserialize(string)` method in derived classes of `BaseMonster`

- Please note that these are class methods of derived classes and you should return a pointer of the base class which points to a newly-constructed instance of derived classes!
- Also, `NovicePlayer` is a class that able to instantiate, so you also need to implement `unserialize()` for `NovicePlayer`