Imię i nazwisko:..... **X** czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (5pkt)

Zmień implementację klas związanych ze wzorcem obserwatora, aby działał on poprawnie w sytuacji, gdy obiekt obserwatora konkretnego zostanie usunięty (patrz przykład obok).

```
class Observer {
 public:
   virtual void update() = 0;
   virtual ~Observer(){}
};
class Subject {
public:
  void add(weak_ptr<Observer> o){
    obs_.push_back(o);
  }
  void notify() {
     for(weak_ptr<Observer> o : obs_)
       o.lock()->update();
  }
  virtual ~Subject(){}
private:
  std::vector<weak_ptr<Observer>> obs_;
};
```

```
class ConcreteObs : public Observer,
  public std::enable_shared_from_this<ConcreteObs> {
public:
    //funkcja fabryczna
  static shared_ptr<ConcreteObs> create(shared_ptr<Subject> sub) {
     shared_ptr<ConcreteObs> ptr(new ConcreteObs(sub));
     ptr->init();
  }
  ConcreteObs(shared_ptr<Subject> sub) : sub_(sub) { }
  void init() {
    me_ = shared_from_this();
    sub_->add(me_);
  }
  virtual ~ConcreteObs() { }
  virtual void update() { }
private:
  shared_ptr<Subject> sub_;
  weak_ptr<ConcreteObs> me_;
};
void zad1() {
   shared_ptr<Subject> subject(new Subject);
   shared_ptr<ConcreteObs> obs1 = ConcreteObs::create(subject);
   shared_ptr<ConcreteObs> obs2 = ConcreteObs::create(subject);
   obs1.reset();
   subject->notify();
}
```

Zadanie 2 (3pkt)

Liczba liter Twojego nazwiska LICZBA_LITER= [ ]. Podaj napis generowany przez zad2 [ ]

```
class E : public exception {
public:
  E(int i) : i_(i) {}
  int i_;
};
struct F {
  F(int i) : i_(i) { }
  virtual ~F() { cout << i_ << '␣';}
  int i_;
};
```

```
typedef unique_ptr<F> PF;
void f(PF&& pf) {
  if(pf->i_ > 2)
    f( PF(new F(pf->i_ - 2)));
  else
    throw E(pf->i_);
}
void f(const PF& pf) {
  if( pf->i_ > 1)
    f( PF(new F(pf->i_ / 2)));
}
```

```
void zad2() {
  try {
    PF pb(new F(LICZBA_LITER - 1));
    f(pb);
  } catch(E& e){
    cout << e.i_;
  }
  cout << endl;
}
```

Zadanie 3 (5pkt)

```
class BaseException : public exception {
public:
  virtual void accept(Visitor& v) = 0;
};
class Visitor {
public:
  virtual ~Visitor() {}
  virtual void visit(NetworkError&) = 0;
  virtual void visit(UserBreak&) = 0;
  virtual void visit(BadData&) = 0;
};
```

Popraw wydajność obsługi wyjątków w funkcji zad3().

```
class NetworkError : public BaseException {
public: virtual void accept(Visitor& v) { v.visit(*this); }
};
class UserBreak : public BaseException {
public: virtual void accept(Visitor& v) { v.visit(*this); }
};
class BadData : public BaseException {
public: virtual void accept(Visitor& v) { v.visit(*this); }
};
```

```
  void zad3() {
    try {

      //kod moze zglosic wyjatek z hierarchii BaseException

    } catch(NetworkError&) {
       cerr << "Network␣Error" << endl;
    } catch(UserBreak&) {
         cerr << "User␣Break" << endl;
    } catch(BadData&) {
       cerr << "Bad␣Data" << endl;
    }
  }
```

Zadanie 4 (5pkt)

`CFoo` to proxy zapewniające kopiowanie przy pisaniu dla klasy `Foo`. Dodaj implementację metod generowanych domyślnie: konstruktor kopiujący, konstruktor przenoszący, operator przypisania, operator przypisania przenoszący, destruktor.

```
struct Foo {
  Foo(int v) : val_(v), counter_(1) {}
  ~Foo() {}
  int val_;
  int counter_;
};
```

```
class CFoo {
public:
  CFoo(int v) : foo_(new Foo(v) ) {}
  void set(int v) {
    if(foo_->counter_ == 1) {
      foo_->val_ = v;
    }
    else {
      --foo_->counter_;
      foo_ = new Foo(v);
    }
  }
  int get() const { return foo_->val_; }
private:
  void join(Foo* f) {
    foo_ = f;
    ++foo_->counter_;
  }
  void unjoin() {
    --foo_->counter_;
     if(foo_->counter_ == 0)
       delete foo_;
  }
  Foo* foo_;
};
```

Zadanie 5 (6pkt)

Obiekty `Tank`, które dostarczają metodę `fire`, można utworzyć w 8 wariantach: normalny celownik lub ulepszony celownik, z typową amunicją lub ulepszoną amunicją, z typową armatą lub ulepszoną armatą. Każda z cech (celownik, amunicja, armata) niezależnie od siebie modyfikuje metodę `fire`. Zaproponuj klasy konkretne, do reprezentacji takich obiektów.

```
class Tank {
public:
   virtual void fire() = 0;
};
typedef shared_ptr<Tank> PTank;
```

Pytanie 1 (1pkt)

Dlaczego używa się różnych języków programowania do tworzenia tej samej aplikacji ?

Uwagi do prowadzącego: