א. מימוש:

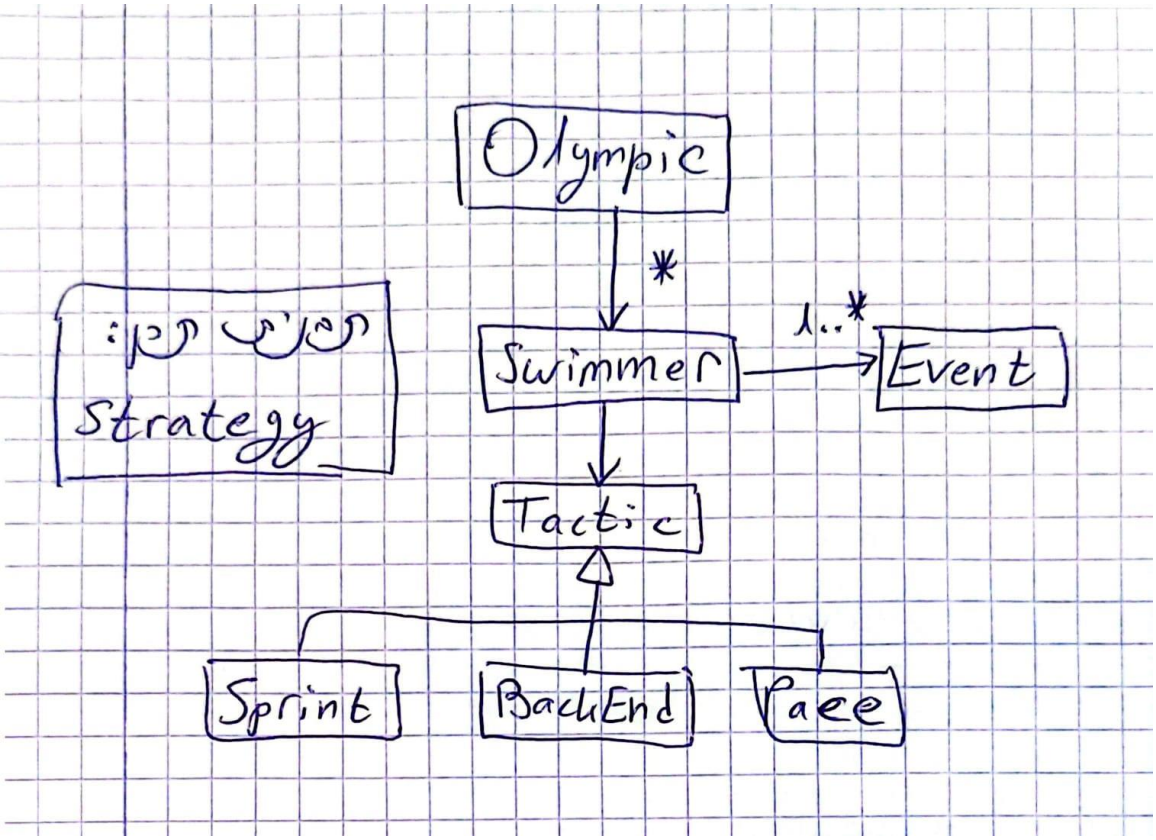
```
class Sprint : public Tactic {
public:
    Sprint() : Tactic() {}
    ~Sprint() override = default;
    shared_ptr<Tactic> clone() const override {
        return shared_ptr<Tactic>(new Sprint());
    }
    float swim(int distance) const override {
        return sqrt(distance)/10;
    }
};

class BackEnd : public Tactic {
public:
    BackEnd() : Tactic() {}
    ~BackEnd() override = default;
    shared_ptr<Tactic> clone() const override {
        return shared_ptr<Tactic>(new BackEnd());
    }
    float swim(int distance) const override {
        return 1-pow((distance-2600)/2600, 2);
    }
};

class Pace : public Tactic {
public:
    Pace() : Tactic() {}
    ~Pace() override = default;
    shared_ptr<Tactic> clone() const override {
        return shared_ptr<Tactic>(new Pace());
    }
    float swim(int distance) const override {
        return distance/1500;
    }
};
```

ב. מימוש:

```
class Event {
    int m_distance;
    Stroke m_stroke;

    static const int DEFAULT_DIST = 50;
    static const Stroke DEFAULT_STROKE = Free;

    bool validFlyBackBreastDist() const {
        return (m_distance == 100 || m_distance == 200);
    }
    bool validFreeDist() const {
        return (m_distance == 50 || m_distance == 100
             || m_distance == 200 || m_distance == 400
             || m_distance == 800 || m_distance == 1500);
    }
    bool validIMDist() const {
        return (m_distance == 200 || m_distance == 400);
    }

public:
    Event(int distance =DEFAULT_DIST, Stroke stroke =DEFAULT_STROKE) :
        m_distance(distance), m_stroke(stroke)
    {
        switch (m_stroke)
        {
        case Fly:
        case Back:
        case Breast:
            if (!validFlyBackBreastDist()) {
                throw InvalidEvent("Unofficial event");
            }
            break;
        case Free:
            if (!validFreeDist()) {
                throw InvalidEvent("Unofficial event");
            }
            break;
        case IM:
            if (!validIMDist()) {
                throw InvalidEvent("Unofficial event");
            }
            break;
    }
```

```
    Stroke getStroke() const {
        return m_stroke;
    }
    int getDistance() const {
        return m_distance;
    }
};

class InvalidEvent : public runtime_error {
public:
    InvalidEvent(const string& err_msg) : runtime_error(err_msg) {}
};
```

ג. תרשים:



*התקבל גם תרשים שאינו כלל את המחלקה Olympics.

ד. מימוש:

```
class Swimmer {
    const float m_energy;
    shared_ptr<Tactic> m_tactic;
    set<Event> m_events;

    bool over100(const Event& event) const {
        return event.getDistance() > 100;
    }

    int eventsOver100() const {
        int res = 0;
        for (const Event& event : m_events) {
            if(over100(event)) {
                res++;
            }
        }
        return res;
    }

    bool isSprinter() const {
        return dynamic_cast<Sprint*>(m_tactic.get()) != nullptr;
    }

    bool eventSuitsSwimmer(const Event& event) const {
        return !isSprinter() || !over100(event);
    }

public:
    Swimmer(float energy, const Tactic& tactic, set<Event> events) :
        m_energy(energy), m_tactic(tactic.clone()), m_events(events)
    {
        if (m_energy <= 0) {
            throw InvalidSwimmer("Energy is too low");
        }
        if (m_events.size() < 1) {
            throw InvalidSwimmer("Too few events");
        }
        if (isSprinter() && eventsOver100() > 0) {
            throw InvalidSwimmer("Sprinter can't swim more than 100m");
        }
    }

    bool canSwim(const Event& event) const {
        return m_events.count(event) == 1;
    }
```

```
    float swim(const Event& event) const {
        return m_energy * m_tactic->swim(event.getDistance());
    }

    bool addEvent(const Event& event) {
        if (!canSwim(event) && eventSuitsSwimmer(event)) {
            m_events.insert(event);
        }
    }
};

class InvalidSwimmer : public runtime_error {
public:
    InvalidSwimmer(const string& err_msg) : runtime_error(err_msg) {}
};
```

ה. מימוש:

```
set<Swimmer> Olympics::Race(const Event& event) const {
    map<Swimmer, float> resultsTable;
    for (const Swimmer& swimmer : m_swimmers) {
        resultsTable[swimmer] = swimmer.swim(event.getDistance());
    }

    float best_result = resultsTable[*(m_swimmers.begin())];
    for (const Swimmer& swimmer : m_swimmers) {
        if (resultsTable[swimmer] > best_result) {
            best_result = resultsTable[swimmer];
        }
    }

    set<Swimmer> winners;
    for (const Swimmer& swimmer : m_swimmers) {
        if (resultsTable[swimmer] == best_result) {
            winners.insert(swimmer);
        }
    }

    return winners;
}
```