Informatyka 2 **Kolokwium II – Grupa α**

Imię i nazwisko: Nr indeksu: Data: 2018-01-19

Grupa (pn / wt / śr / czw / pt , godzina):

Uwagi

- Na niniejszym arkuszu umieszczaj końcowe rozwiązania. Miejscem na eksperymenty jest brudnopis.

Zadanie 1 [3 p.]

Funkcjonalność sprawdzania uprawnień użytkownika (logika metody `UserSettingService::hasPermissions()`) będzie wykorzystywana również w innych klasach tworzonego systemu, przy czym klasy te nie potrzebują znajomości pozostałego fragmentu klasy `UserSettingService`. Mając to na uwadze, dokonaj takiej modyfikacji poniższego kodu, aby spełniał on zasady SRP/ISP:

```
class User { /* ... */ };

class UserSettingService {
  public:
    void changeEmail(User user) {
      if (hasPermissions(user)) {
        // Grant option to change.
      }
    }
    bool hasPermissions(User user) {
      // Verify if the user has access
          rights.
    }
    // ...
};
```

Miejsce na rozwiązanie:

Zadanie 2 [3 p.]

Zmodyfikuj poniższy kod tak, aby spełniał zasadę OCP:

```
class Worker { /* ...*/ };

class Programmer : public Worker {
    public:
      void code() { /* ... */ }
};

class Tester : public Worker {
    public:
      void test() { /* ... */ }
};
```

```
class ProjectManagement {
  public:
    void processCode(Worker& worker) {

      if (typeid(worker) == typeid(Programmer)) {
          static_cast<Programmer&>(worker).code();
      } else if (typeid(worker) == typeid(Tester)) {
          static_cast<Tester&>(worker).test();
      }
    }
};
```

Zadanie 3 [3 p.]

Wyjaśnij dlaczego poniższy kod nie spełnia zasady LSP oraz zaproponuj (opisz) sposób jego poprawy tak, aby spełniał tę zasadę:

```
class Tile { /* ... */ };

class Board {
  public:
    Tile getTile(int x, int y);
  /* ... */
};

class ThreeDBoard : public Board {
  public:
    Tile getTile(int x, int y, int z);
  /* ... */
};
```

Zadanie 4 [3 p.]

Zmodyfikuj poniższy kod tak, aby spełniał zasady SRP, OCI i ISP:

```
class InterfaceIO {
  public:
    virtual bool canReceive() = 0;
    virtual void receive() = 0;
    virtual void send() = 0;
    virtual ~InterfaceIO() = default;
};

class InterfaceManager {
  public:
    void processInterface(InterfaceIO& interface) {
      if (interface.canReceive()) {
        interface.receive();
      }
    }
};
```

```
class Receiver : public InterfaceIO {
    public:
      bool canReceive() override { return true; }
      void receive() override { /* ... */ }
      void send() override {
          throw std::logic_error("can't send");
      }
};

class Sender : public InterfaceIO {
    public:
      bool canReceive() override { return false; }
      void receive() override {
          throw std::logic_error("can't receive");
      }
      void send() override { /* ... */ }
};
```