# Напредни Пајтон

Принцип SOLID - Напредни ООР

# Увод

- Принципи фокусирани на објектно-орјентисано програмирање. У питању су *best-practices*
- Uncle Bob – Robert Martin
  - Увео принципе
- Michael C. Feathers
  - Сковао кованицу
- Дизајн, одржавање и проширивање апликације постаје најједноставније, а багови се своде на минимум

# S O L I D

- S – Single Responsibility Principle
  - Јединична цјелина кода (модул, класа, функција) има само једну одговорност
- O – Open/Close Principle
  - Цјелина је ОТВОРЕНА за надоградњу, а ЗАТВОРЕНА за измјене
- L – Liskov Principle of Substitution
  - Ако референцу на родитељски објекат замијенимо са референцом на дијете-објекат, програм неће јавити грешку
- I – Interface Segregation Principle
  - Нико не смије да зависи од метода које не позива
- D – Dependency Inversion Principle
  - Класе вишег нивоа не смију зависити од класа нишег нивоа
  - Апстракције не смију зависити од детаља, него обратно

# Принцип једне одговорности

- Разне су дефиниције, односно различит је фокус

The Single Responsibility Principle (SRP) states that each software module should have one and only one reason to change.

Single Responsibility Principle states that

"A class should have one and only one reason to change."

The main idea of this concept is: all pieces of software must have only a single responsibility.

- Избјегавати да се **све одради на једном мјесту!**
- Најоптималнији је фокус на **ФУНКЦИЈУ**

# S O L I D

- S – Single Responsibility Principle
  - Јединична цјелина кода (модул, класа, функција) има само једну одговорност
- O – Open/Close Principle
  - Цјелина је ОТВОРЕНА за надоградњу, а ЗАТВОРЕНА за измјене
- L – Liskov Principle of Substitution
  - Ако референцу на родитељски објекат замијенимо са референцом на дијете-објекат, програм неће јавити грешку
- I – Interface Segregation Principle
  - Нико не смије да зависи од метода које не позива
- D – Dependency Inversion Principle
  - Класе вишег нивоа не смију зависити од класа нишег нивоа
  - Апстракције не смију зависити од детаља, него обратно

# Отворено, али затворено

- Одговорити на питања:
  - Шта?
    - Најстабилнији дио кода
  - Како?
    - Најкомплекснији дио кода
    - Постаје посебна функција
  - Зашто?
    - Дио који дијели један *КАКО* од другог
    - Постаје особина
- Подијелити код у складу са одговорима
- Могуће је да ће се ово слити у неки од дизајн шаблона .

# S O L I D

- S – Single Responsibility Principle
  - Јединична цјелина кода (модул, класа, функција) има само једну одговорност
- O – Open/Close Principle
  - Цјелина је ОТВОРЕНА за надоградњу, а ЗАТВОРЕНА за измјене
- L – Liskov Principle of Substitution
  - Ако референцу на родитељски објекат замијенимо са референцом на дијете-објекат, програм неће јавити грешку
- I – Interface Segregation Principle
  - Нико не смије да зависи од метода које не позива
- D – Dependency Inversion Principle
  - Класе вишег нивоа не смију зависити од класа нишег нивоа
  - Апстракције не смију зависити од детаља, него обратно

# Сигурна замјена референци госпође Лисков

- У питању је ИНТЕГРИТЕТ који можда никад неће бити нарушен!
- У глобалу изгледа да улазни типови и повратни тип метода мора бити исти, али је много шире од тога
- Пошто је Пајтон веома либералан језик када су у питању типови ово је још битнији принцип који треба да се поштује.
- Рјешава се:
  - Додатним провјерама
  - Избацивањем особина или понашања из родитељских класа
  - Апстрактовањем родитељске класе и метода
  - MethodNotImplementedException

# S O L I D

- S – Single Responsibility Principle
  - Јединична цјелина кода (модул, класа, функција) има само једну одговорност
- O – Open/Close Principle
  - Цјелина је ОТВОРЕНА за надоградњу, а ЗАТВОРЕНА за измјене
- L – Liskov Principle of Substitution
  - Ако референцу на родитељски објекат замијенимо са референцом на дијете-објекат, програм неће јавити грешку
- I – Interface Segregation Principle
  - Нико не смије да зависи од метода које не позива
- D – Dependency Inversion Principle
  - Класе вишег нивоа не смију зависити од класа нишег нивоа
  - Апстракције не смију зависити од детаља, него обратно

# Сегрегација интерфејса

- Проблем који се овим избјегава се дешава када превише особина и понашања издефинишемо у родитељским класама
- ПОСЉЕДИЦА: Прављење мањих интерфејса
- Можда једино мјесто гдје има смисла радити вишеструка насљеђивања, али искључиво као посљедица чињенице да у Пајтону не постоје интерфејси на начин на који постоје у другим језицима.

# S O L I D

- S – Single Responsibility Principle
  - Јединична цјелина кода (модул, класа, функција) има само једну одговорност
- O – Open/Close Principle
  - Цјелина је ОТВОРЕНА за надоградњу, а ЗАТВОРЕНА за измјене
- L – Liskov Principle of Substitution
  - Ако референцу на родитељски објекат замијенимо са референцом на дијете-објекат, програм неће јавити грешку
- I – Interface Segregation Principle
  - Нико не смије да зависи од метода које не позива
- D – Dependency Inversion Principle
  - Класе вишег нивоа не смију зависити од класа нишег нивоа
  - Апстракције не смију зависити од детаља, него обратно

# Инверзија зависности

- Ако су задовољени OCP и LPS онда је већ имплементиран и овај принцип, тако да није обавезан као остали. Међутим, његово рјешавање може да поједностави рјешавање OCP и LPS
- Проблем се појављује углавном касније када приликом проширења система, испостави се да се И виши нивои морају мијењати заједно са нижим

# ГИТ за СОЛИД?

- Репозиториј са још примјера и рјешења
  - Могуће је да код није у потпуности тачан.