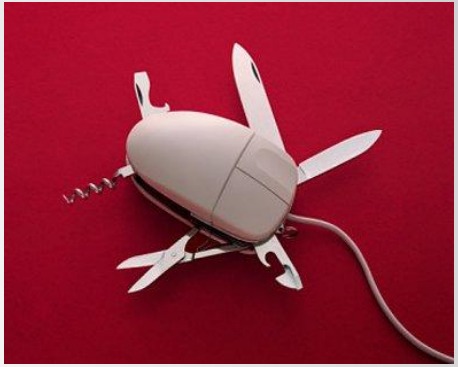
ООП

ООП

- композиция и наследование
- абстрактный класс и интерфейс
- интерфейсы и типы
- внутренние и вложенные классы
- анонимные и локальные классы
- шаблон Стратегия

Объектно-ориентированное программирование

Наследование - самый сильный способ связи между классами. Дочерний класс является тем же, что и класс родитель (IS-A). Абстрагируемся от реализации используя базовый класс. Базовый класс определяет поведение дочернего.



Композиция - отношение HAS-A. Класс обладает свойствами своих составных частей.Абстрагируемся от реализации, использую интерфейс. Можем динамически менять поведение класса.

Инкапсуляция и наследование

Наследование - переиспользование кода

Инкапсуляция - сокрытие и контроль внутреннего состояния

Access Levels

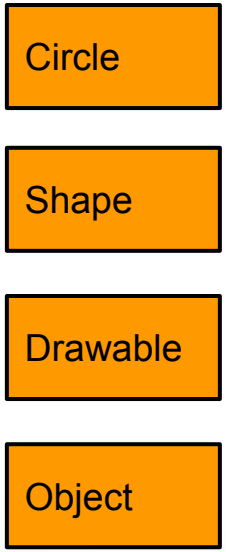
Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

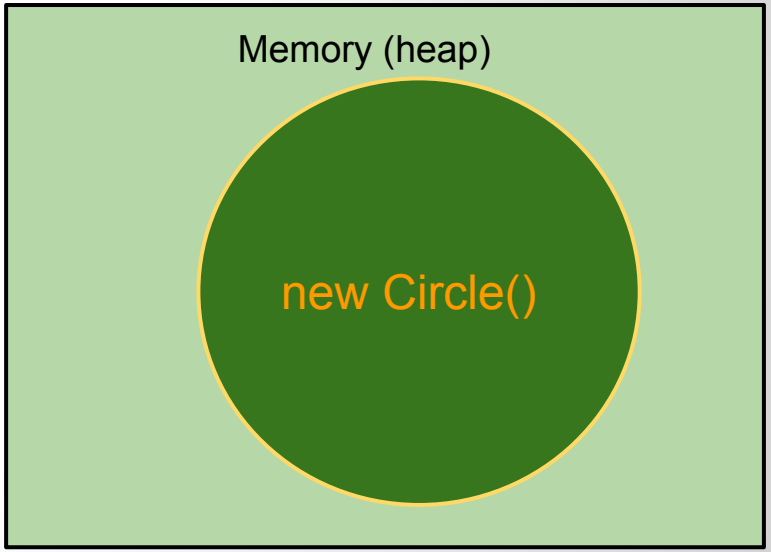
Ссылки и объекты

```
interface Drawable {
    void draw();
}

abstract class Shape
implements Drawable {
    int x,y;
}

class Circle extends Shape {
    int r;
    void draw() {
        …
    }
}
```





Полиморфизм

- Дочерний класс может быть использован везде, где используется родительский
- Если дочерний класс приведен к родительскому, то доступны только методы родительского класса
- Вызывается реализация из дочернего класса (@Override)

Связывание

Раннее связывание (этап компиляции)

- поиск подходящей сигнатуры (Overloading)

Позднее связывание (этап исполнения)

- поиск подходящей реализации (Overriding)

Dependency Inversion

1. Модули более высокого уровня не должны зависеть от модулей более низкого уровня. И те, и другие должны зависеть от абстракций.

2. Абстракция не должна зависеть от деталей реализации. Реализация должна зависеть от абстракции.

Single Responsibility

Объект должен иметь одну обязанность и она должна быть инкапсулирована внутри объекта

```
Controller {
    DataSource ds;
}
```

Контроллер видит только интерфейс (зависит от абстракции)

```
interface DataSource {
  List getUsers();
}
```

Реализация DataSource не знает ничего о контроллере

Интерфейс DataStore ничего не знает о своей реализации

```
DbDataSource {
....
}
```

Dependency Injection

Dependency Injection – внедрение зависимостей

необходимые объекты поставляются извне

- через конструктор
- через методы get()/set()

	Абстрактный класс	**Интерфейс**
использование	extends, можно наследоваться только от одного класса	implements, можно реализовывать несколько интерфейсов
реализация	может иметь	не имеет
модификаторы доступа	любые (но для abstract методов только public/protected)	public (by default)
переменные	любые	final (by default)
конструктор	может иметь	не имеет

Интерфейсы и типы

- Интерфейсы позволяют строить гибкую архитектуру
- Реализацию всегда можно подменить
- Интерфейс - это контракт
- Взаимодействие между компонентами системы

Архитектура

- Инкапсуляция - сокрытие внутреннего состояния
- Coupling - информация о внутренней реализации других компонентов. Чем меньше знаем, тем лучше
- Cohesion - разделенность системы на отдельные компоненты. Каждый компонент должен представлять одну сущность и решать одну задачу.

Внутренний и вложенный класс

- Логическая группировка
- Инкапсуляция
- Читаемость

Анонимный класс

- Не имеет имени
- Доступ только к final полям

Шаблон Стратегия

- Подмена алгоритма
- Аналог callback

Обработка коллекций

unmodifiable collections (создает копию)

Основные операции

- filter (фильтрация по условию)
- reduce (операции с накоплением)
- map (преобразование элементов)