# Программирование на языке Java. Часть 2. Лекция 3

Курс для самостоятельного изучения

# Константы Java

- Часто в коде на языке Java требуется объявить константу

```
public class BoggleBoard {
   /** Размер доски Боггл по умолчанию. */
   public static final int DEFAULT_SIZE = 4;
   ...
}
```

- Стандартные соглашения для констант Java:
  - Имя задается по схеме ALL_CAPS (слова из заглавных букв разделенные подчеркиванием)
  - Используются модификаторы доступа public static final
  - (или, если нужно private / protected)

# Ключевое слово static

- Члены класса можно объявить статическими
- Они относятся к классу, а не к определенному объекту
- Статические поля имеют только одну копию значения
- Пример:

```
public class CommandPrompt {
   public static final String PROMPT =
        "Введите команду: ";
   ...
}
```

- PROMPT это объект, но он не принадлежит какому либо экземпляру класса CommandPrompt
- Есть только одно значение, и весь код совместно использует это единственное значение
- Память используется более эффективно чем в случае, если хранить значения в полях экземпляров класса

# Инициализация статических полей

- Когда инициализируются статические поля?

```
public class CommandPrompt {
   public static final String PROMPT =
       "Введите команду: ";
   ...
}
```

- ВМ инициализирует класс когда тип первый раз используется в коде.
  - Определение класса находится через classpath, и затем проверяется
    - т.е. проверяется, что все инструкции правильные; инструкции перехода указывают на правильные адреса и т.п.
  - Любая ссылка на другой тип может быть проверена и создана
    - (для этого может потребоваться. конечно, загрузка дополнительных классов)
  - Наконец, инициализируются статические поля класса

# Инициализация статических полей (2)

- Статические поля инициализируются в конце процесса загрузки класса
- Иногда, невозможно проинициализировать статическое поле одной строкой кода

```
public class NoiseGenerator {
    public static final Vector3f[] noiseVectors =
            new Vector3f[1024];
    ...
}
```

  - Нужно также проинициализировать вектор отсчетов шума случайными значениями
  - Конечно это нельзя сделать одной строкой!
- Как надо проинициализировать это статическое поле?

# Инициализация статических полей (3)

- Классы могут иметь статические инициализаторы:

```
public class NoiseGenerator {
    public static final Vector3f[] noiseVectors =
         new Vector3f[1024];
    static {
         for (int i = 0; i < noiseVectors.length; i++) {
                 noiseVectors[i] = new Vector3f();
                 ... // Инициализация вектора
         }
    }
    ...
}
```

- Статические инициализаторы не могут генерировать проверяемые исключения!
- Инициализация статических полей, и исполнение инициализаторов производится в порядки расположения в файле исходного кода
- Инициализация статических полей из нескольких потоков автоматически синхронизируется

# Ключевое слово final

- Переменные Java могут быть объявлены final
  - Значение такой переменной можно присвоить только один раз.
- Часто используется для полей-констант классов и экземпляров

```
public class CommandPrompt {
   public static final String PROMPT =
        "Введите команду: ";
   ...
}
```

  - PROMPT можно изменить только раз и это значение остается навсегда
- Поля final обычно получают значение прямо там где объявляются, но в Java так делать не обязательно!
  - final поля экземпляра должны получить значение до завершения кода конструктора
  - Статические final поля (поля класса) должны инициализироваться статическим инициализатором

# Ключевое слово final (2)

- Иногда final используется с локальными переменными и аргументами методов
  - Запрещает переприсвоение значений переменным которые не должны изменяться
  - Используется для снижения вероятности появления ошибок
  - Польза этой технологии не всегда очевидна… ☺
- Пример:

```
int findWord(String w, final ArrayList<String> words) {
    int i = 0;
    for (String s : words) {
         if (s.equals(w)) return i;
         i++;
    }
    return -1;
}
```

- Что можно сделать со словами?
  - Слово не может ссылаться а что то еще
  - Увеличиваем корректность нашего метода (слегка)

# Ключевое слово final (3)

- Пример:

```
int findWord(String w, final ArrayList<String> words) {
    int i = 0;
    for (String s : words) {
         if (s.equals(w)) return i;
         i++;
    }
    return -1;
}
```

- Что можно сделать со словами?
  - Мы можем вызывать разные методы для слов…
  - Мы можем вызывать мутаторы для слов!
    - `words.add("yo' mama!");`
    - `words.clear();`
- final всего лишь запрещает повторное присвоение значений переменной
- Объявление переменной final не так уж много дает…

# final и const

- Ключевое слово final в Java не похоже на const в C++
  - (и в Java нет эквивалента C++ const)
- Вы возможно видели проекты, в которых использовалось final для аргументов методов и локальных переменных…
  - Надо иметь ввиду ограниченность этой технологии
- Если вам действительно нужно запретить изменения:
- Создайте класс без мутаторов!
  - (добавьте если нужно, дочерний класс с мутаторами)
  - Классы Java String, Integer, и т.п. все неизменяемые
- Или воспользуйтесь Collections.unmodifiableList(List), и пр.
  - Обеспечивает просмотр без возможности изменения других коллекций
  - Исходная коллекция по прежнему может редактироваться, но методам которые с ней работают можно передать ее неизменяемый вид

# Вернемся к константам Java

- Стандартный набор модификаторов для констант

```
public class BoggleBoard {
/** Размер доски Боггл по умолчанию. **/
public static final int DEFAULT_SIZE = 4;
...
}
```

- Для простых констант это рекомендуемый набор
- Если константа объект, это повышает эффективность использования памяти
- Есть еще два часто встречающихся способа использования констант
  - Оба не на столько хорошие ☺

# Интерфейсы и константы

- Интерфейсы могут содержать публичные методы и константы!
  - Константы достаточно объявить static final, так как все члены интерфейса автоматически имеют модификатор public
- Когда в пакете используется констант их обычно помещают в отдельный “интерфейс констант”
  - В таком интерфейсе есть только константы, а методы отсутствуют
- В Java API есть много таких примеров
  - Интерфейс javax.swing.SwingConstants
    - Содержит константы выравнивания LEFT, CENTER, RIGHT
- Многие классы Swing “реализуют” SwingConstants, поэтому они могут использовать эти константы в своих реализациях
  - При этом не нужно добавлять методы; у SwingConstants их нет!

# Джошуа Блох и интерфейсы констант

- Интерфейс это тип в Java
- Интерфейсы задают набор поведений, которые реализуются объектами
- Если класс реализует интерфейс:
  - интерфейс говорит о том, что клиенты класса могут делать с объектами этого типа!
  - Другой код может ссылаться на объект используя тип интерфйеса
- Интерфейсы констант нарушают этот принцип
  - Т.е., SwingConstants вовсе не содержит описание поведения!
  - Но мы можем, например, написать такой странный код:

```
SwingConstants c = new JButton("Это странно");
```

- Нельзя вызвать какой либо метод, потому что ни один не объявлен!

# Лучшее решение: служебные классы констант

- Если вам надо собрать вместе большое количество констант:
  - Поместите их в служебный класс, и сделайте экземпляр этого классcan
    - Класс должен иметь private конструктор по умолчанию
  - Сделайте для констант поля public static final
- Мораль:
- Если в Java API используются какие то приемы программирования, это автоматически не означает что вы должны им следовать. ☺

# Простая нумерация

- Константы часто используются для нумерации элементов

```
/** Колода карт. */
public class Card {
public static final int SPADES = 1;
public static final int HEARTS = 2;
public static final int CLUBS = 3;
public static final int DIAMONDS = 4;
...
}
```

- Проблемы?
  - Небезопасное использование типов:

```
public class Card {
...
void setSuit(int suit);
}
```

- Можно случайно перепутать номера, или указать недопустимое значение!

# Нумерация с безопасным использованием типов

- Показанная выше нумерация повышает вероятность ошибок
- Лучше использовать: “нумерацию с безопасным использованием типов”
  - Создайте класс для перечисления
  - Создайте уникальный объект для каждого перечисляемого значения

```
public class Suit {
/** Только Suit может вызвать свой конструктор. */
private Suit() { }
public static final Suit SPADES = new Suit();
public static final Suit HEARTS = new Suit();
public static final Suit CLUBS = new Suit();
public static final Suit DIAMONDS = new Suit();
}
```

- Можно добавить другие поля к перечисляемому значению, например, name, id, и пр.

# Нумерация с безопасным использованием типов (2)

- “Нумерация с безопасным использованием типов” это очень полезный прием, но он также требует много кода
  - Главное что каждое из перечисляемых значений уникально в виртуальной машине Javan
- С объектами нельзя использовать оператор switch:

```
Card c = ... ;
switch (c.getSuit()) {
   case Suit.SPADES:
   ...
}
```

  - Этот код не будет компилироваться, если применяется подход с безопасным использованием типов!
  - Но будет компилироваться, если масти представлены целыми числами, однако, как мы помним, этот вариант имеет  большие проблемы

# Типы enum в Java 1.5

- В версии Java 1.5 введена поддержка перечисляемых типов (enum)
  - Правила использования очень простые …
  - Реализация иногда бывает запутанной…
  - и нам бы хотелось иметь поддержку языка (т.е. switch)
- Переделаем класс Suit так чтобы он поддерживал перечисляемы типы:

```
public enum Suit {
SPADES,
HEARTS,
CLUBS,
DIAMONDS
}
```

- Комментарии Javadoc можно добавлять к типу и каждому элементу перечисления

# Типы enum в Java 1.5 (2)

- Оператор switch можно использовать с перечисляемыми типами:

```
Card c = ... ;
switch (c.getSuit()) {
   case SPADES:
     ...
}
```

- Перечисляемые типы в Java получают автоматически метод toString() и другие полезные методы класса Object

```
System.out.println(c.getSuit());
→SPADES
```

- К значениям перечисляемого типа можно обращаться как к элементам массива

```
for (int val = 1; val <= 13; val++)
   for (Suit s : Suit.values)
     deck.add(new Card(val, s));
```

# Расширение перечисляемых типов

- Перечисляемые типы Java это классы
  - К перечисляемому типу можно добавить поля и методы
- Пример:

```
public enum ChessPiece {
  KING (200), // Произвольное значение для короля
  QUEEN (9),
  ROOK (5),
  BISHOP(3),
  KNIGHT(3),
  PAWN (1); // Обратите внимание на точку с запятой!
  private final int value; // Вес фигуры
  ChessPiece(int value) { this.value = value; }
  public int value() { return value; }
}
```

# Вложенная нумерация

- Перечисляемый тип можно объявить внутри другого класса

```
public class Card {
   public enum Suit {
    SPADES, HEARTS, CLUBS, DIAMONDS
   }
   public Card(int value, Suit suit) {
    ...
   }
}
```

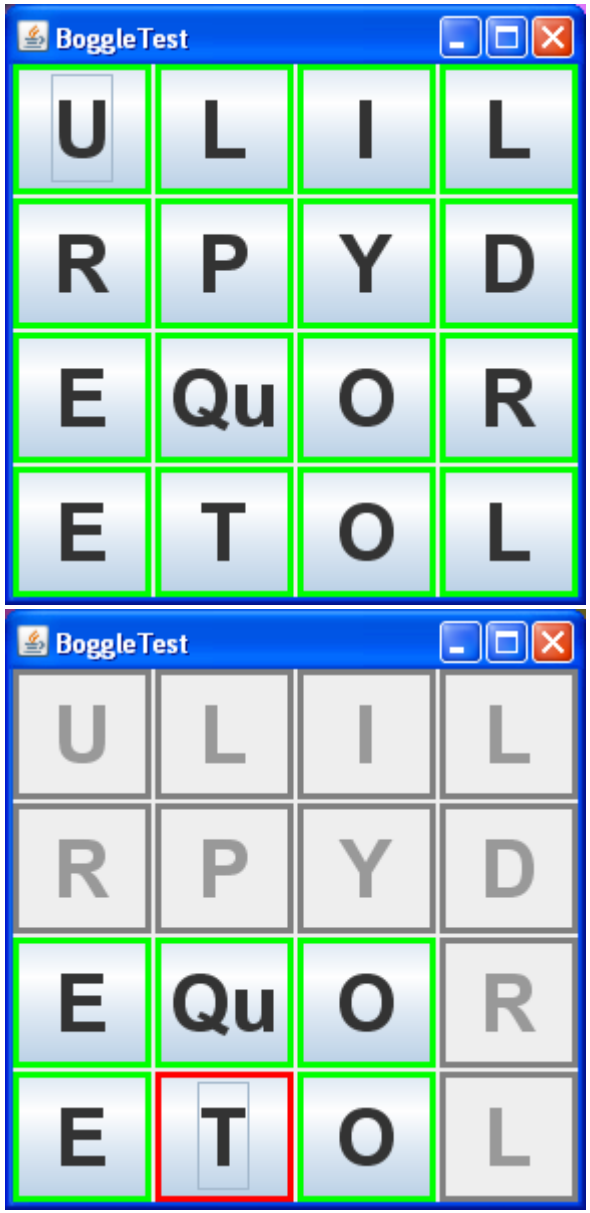
- Код класса Card может ссылаться на перечисляемые значения так:  Suit.SPADES, и т.п.
- Внешний код должен делать так: Card.Suit.SPADES, и т.п.

# Задание на этой неделе

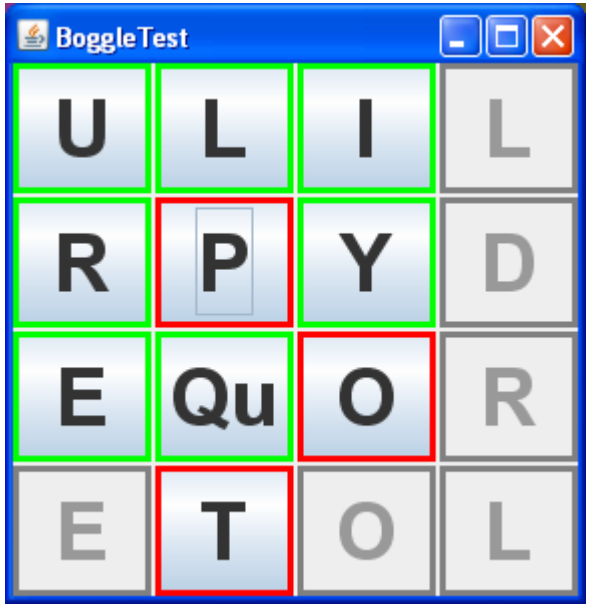
- Начинаем делать пользовательский интерфейс игры Боггл
  - Начинаем с класса который рисует доску Боггл и позволяет иголкам вводить найденные ими слова
- Пользователь должен иметь визуальные подсказки
  - Используем большой легко читаемый шрифт
  - Используем “активное” состояние и цвет рамки для того чтобы указать какую букву можно выбрать на следующем ходе
- Пользовательский интерфейс также должен иметь метод возвращающий текущее выбранное слово

# Пример пользовательского интерфейса

- Поле игры Боггл состоит из клеток (кнопок)
- По цвету рамок кнопок можно определить, какие из кнопок можно выбрать
- Когда игрок выбирает букву, ее рамка становится красной
- Можно использовать, только кнопки расположенные рядом с последней выбранной кнопкой

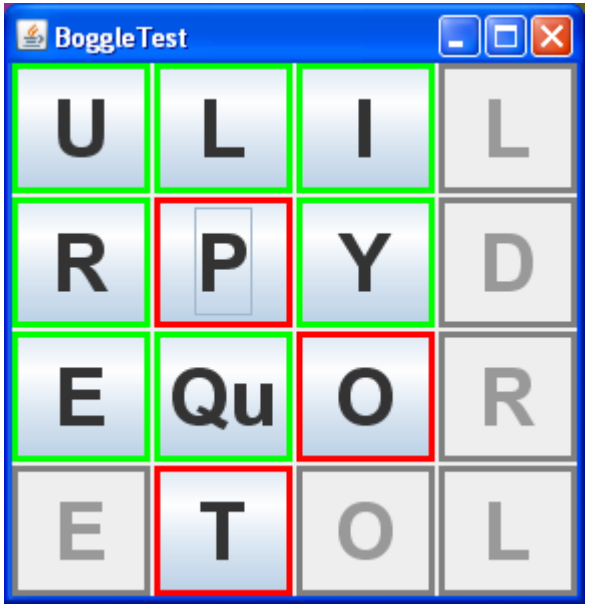


- После того как буквы выбраны слово выводится красным цветом
- Само слово составлено из букв кнопок
- “Доступные буквы” всегда определяются по последней выбранной букве
- Уже выбранные буквы исключаются!



# Общий подход к решению задачи

- Не изобретайте колесо!
- В Swing уже есть готовые кнопки и панели
  - Просто модифицируйте их поведение!
- Создайте класс потомок JButton который отображает клетку игры Боггл так как это требуется в задании
  - Управляйте состоянием кнопки, внешним видом, значением клетки, и пр.
- Создайте класс потомок JPanel который отображает всю доску Боггл
  - Методы для подготовки доски к игре, и получения текущего слова
  - Обработчики событий от кнопок для обновления их внешнего вида

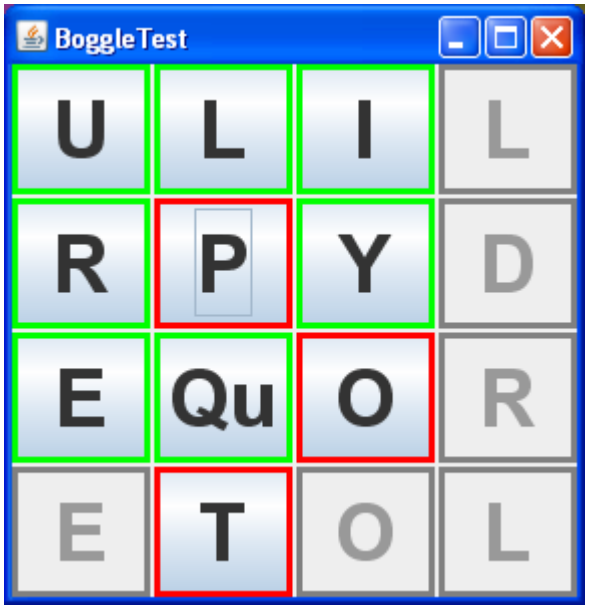


# Внешний вид компонентов Swing

- Все компоненты Swing наследуются от javax.swing.JComponent
  - Он содержит общий функционал для всех компонентов
  - Пользовательские компоненты, которые сами прорисовывают свой внешний вид также наследуются от JComponent
- Есть много способов поменять вид JComponent
  - Задать всплывающую подсказку, добавить одну или несколько рамок, изменить цвет фона/текста, изменить видкурсора, шрифт, и т.д.
- Компонент можно также активировать/деактивировать (enable/disable)
  - Неактивный компонент не получает пользовательский ввод
  - Пользовательский интерфейс прорисовывает такой элемент в серых тонах
  - Используйте методы setEnabled(boolean) и isEnabled()

# Именование компонентов Swing

- Соглашение о именах компонентов Swing
- Все компоненты Swing наследуются от JComponent
  - Это Swing аналог типа Java AWT Component
- Имена всех компонентов Swing начинаются с буквы “J”
- За исключением случаев, когда это действительно не имеет смысла, следуйте в вашем коде этому соглашению
  - T.e. JBoggleButton, JBoggleBoard



# Компоненты Swing и шрифты

- Шрифт компонентов Swing можно изменить
  - Методы setFont(Font) и getFont()
- Класс java.awt.Font  управляет шрифтами в Java
- Шрифты Java разделяются на две категории:
  - Физические шрифты соответствующие шрифтам установленным на вашем компьютере (например Arial или Helvetica)
  - Логические “встроенные” шрифты которые должны поддерживать все виртуальные машины Java
    - Обычно эти шрифты получаются отображением имен каждого логического шрифта в физический шрифт использующийся по умолчанию в операционной системе
    - Serif, SansSerif, Monospaced, Dialog, и DialogInput

# Компоненты Swing и шрифты (2)

- Легче всего задать шрифт в конструкторе класса Font
  - `Font(String name, int style, int size)`
  - Класс Font имеет константы для всех имен и стилей логических шрифтов

```
// Получаем жирный шрифт размером 20-пунктов не serif
Font f = new Font(Font.SANS_SERIF, Font.BOLD, 20);
```

  - Можно задать другие имена шрифтов, но не гарантируется что они будут доступны!

```
// Получаем наклонный шрифт Times New Roman, 12-пунктов
f = new Font("Times New Roman", Font.ITALIC, 12);
```

  - Если имя шрифта неизвестно Java переходит на логический шрифт “Dialog”
  - Предложение: используйте в конструкторе только логические имена шрифтов

# Компоненты Swing и шрифты (3)

- Чтобы получить список шрифтов в системе используйте: Font[] java.awt.GraphicsEnvironment.getAllFonts()
  - Возвращает массив объектов Font  который содержит все доступные шрифты
  - Все возвращаемы шрифты имеют размер 1-пункт
  - Выглядит это примерно так: . (the dot is “this text is 1-point”)
  - Приложение должно получить шрифты из этих “базовых шрифтов”
- Для того чтобы ваше приложение было масксимально переносимым используйте этот механизм для поиска системных шрифтов
  - Или просто используйте логические шрифты

# Компоненты Swing и рамки

- Компоненты Swing могут иметь рамку
  - Рамка увеличивает размер компонента Swing
- Для установки и получения ссылки на рамку используются методы via setBorder(Border) и getBorder()
- Border это интерфейс определенный в пакете javax.swing.border
  - см. реализации в Java API!
- Есть два способа получить простую рамку:
  - Создать ее самому:

```
Border b = new LineBorder(Color.RED, 3);
```

- Воспользоваться классом javax.swing.BorderFactory

```
Border b = BorderFactory.createLineBorder(Color.RED,
  3);
```

# Ссылки

- Java. Эффективное программирование. Джошуа Блох
- Пункт 17: Используйте интерфейсы только для определения типов
- Пункт 21: Заменяйте конструкцию enum классом