Lesson 26

01.07.2021

```
public class Test1 {
    public static void main(String[] args) {
        byte a = 12;
        byte b = 13;
        byte result = a + b;
        System.out.println(result);
    }
}
```

```
public class Test2 {
    public static void main(String[] args) {
        int x = 011;
        int y = 0xfee;
        int result = x + y;
        System.out.println(result);
    }
}
```

```
public class Test3 {
    public static void main(String[] args) {
        int i = 0;
        int j = 9;
        do {
            i++;
            if (j-- < i++) {
                break;
            }
        } while (i < 5);
        System.out.println(Integer.toString(i)
                + Integer.toString(j));
    }
}
```

```
public class Test4 {
    public static void main(String[] args) {
        int[] x = new int[3];
        System.out.println("x[0] is " + x[0]);
    }
}
```

```
public class Test5 {
    public static void main(String[] args) {
        int x = 5;
        int y = 6;
        if ((y = 1) == x)
            System.out.println(y);
        else
            System.out.println(++y);
    }
}
```

```
public class Test6 {
    public static void main(String[] args) {
        for (int i = 0; i < 3; ) {
            System.out.println("Java");
        }
    }
}
```

Интернационализация

Вопрос интернационализации пользовательского интерфейса - один из важных вопросов при разработке приложения. Для этого недостаточно использовать **Unicode** и перевести на нужный язык все сообщения пользовательского интерфейса. Интернационализация приложения означает нечто большее, чем поддержка Unicode. Дата, время, денежные суммы и даже числа могут по-разному представляться на различных языках.

Широкое распространение получили условные сокращения терминов **интернационализации** и **локализации** приложений i18n и l10n, в которых цифра означает количество символов между первой и последней позицией:

i18n - интернационализация (*internationalization*);
l10n - локализация (*localization*).

Часто встречаются 2 определения:

Интернационализация - это процесс разработки приложения такой структуры, при которой дополнение нового языка не требует перестройки и перекомпиляции (сборки) всего приложения.

Локализация предполагает адаптацию интерфейса приложения под несколько языков. Добавление нового языка может внести определенные сложности в локализацию интерфейса.

Региональные стандарты Locale

Приложение, которое адаптировано для международного рынка, легко определить по возможности выбора языка, для работы с ним. Но профессионально адаптированные приложения могут иметь разные региональные настройки даже для тех стран, в которых используется одинаковый язык. В любом случае команды меню, надписи на кнопках и программные сообщения должны быть переведены на местный язык, возможно с использованием специального национального алфавита. Но существует еще много других более тонких различий, которые касаются форматов представления вещественных чисел (разделители целой и дробной частей, разделителей групп тысяч) и денежных сумм (включение и местоположения денежного знака), а также формата даты (порядок следования и символы разделители дней, месяцев и лет).

Региональный стандарт **Locale** определяет язык. Кроме этого могут быть указаны географическое расположение и вариант языка.

Например, в США используется следующий региональный стандарт:
language=English, location=United States

В Германии региональный стандарт имеет вид :
language=German, location=Germany

В Швейцарии используются четыре официальных языка : немецкий, французский, итальянский и ретороманский. Поэтому немецкие пользователи в Швейцарии, вероятно, захотят использовать следующий региональный стандарт:
language=German, location=Switzerland
В данном случае текст, даты и числа будут форматироваться так же, как и для Германии, но денежные суммы будут отображаться в швейцарских франках, а не в евро. Если задавать только язык, например *language=German*, то особенности конкретной страны (например, формат представления денежных единиц) не будут учтены.

Коды языков

https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D1%8B_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2

Коды стран

https://ru.wikipedia.org/wiki/ISO_3166-1

Предопределенные объекты с региональными установками	Объекты, позволяющие указать язык без указания страны
Locale.CHINA	Locale.CHINESE
Locale.FRANCE	Locale.FRENCH
Locale.GERMANY	Locale.GERMAN
Locale.ITALY	Locale.ITALIAN
Locale.JAPAN	Locale.JAPANESE
Locale.US	Locale.ENGLISH

Помимо вызова конструктора или выбора предопределенных объектов, существует еще два пути получения объектов с региональными настройками. Статический метод *getDefault()* класса **Locale** позволяет определить региональную настройку, которая используется в операционной системе по-умолчанию. Изменить настройку по-умолчанию можно вызвав метод *setDefault ()*. Однако следует помнить, что данный метод воздействует только на Java-программу, а не на операционную систему в целом.

Форматирование числовых значений NumberFormat

В разных странах и регионах используются различные способы представления чисел и денежных сумм. В пакете *java.text* содержатся классы, позволяющие форматировать числа и выполнять разбор их строкового представления. Для форматирования чисел в соответствии с конкретным региональным стандартом необходимо выполнить ряд действий:

Использовать фабричный метод для получения объекта форматирования.
Применить полученный объект форматирования для формирования числа или разбора его строкового представления.
В качестве фабричных методов (*factory method*) используются статические методы *getNumberInstance (), getCurrencyInstance (), getPercentInstance ()* класса **NumberFormat**. Они получают в качестве параметра объект **Locale** и возвращают объекты, предназначенные для форматирования чисел, денежных сумм и значений, выраженных в процентах.

Денежные суммы

Для форматирования денежных сумм используется метод *getCurrencyInstance()* класса **NumberFormat**. Однако этот метод не обеспечивает достаточной гибкости - он возвращает форматированную строку для одной валюты. Допустим, Вы выписываете счет для иностранного потребителя, в котором одни суммы представлены в долларах, а другие в евро. Использование двух приведенных ниже объектов форматирования не является решением задачи.

Для управления форматированием денежных сумм следует использовать класс **Currency**. Для получения объекта *Currency* необходимо передать статическому методу *Currency.getInstance ()* идентификатор валюты. Затем необходимо вызвать метод *setCurrency ()* каждого объекта форматирования.

Идентификаторы валют определены стандартом ISO 4217.

Форматирование даты и времени DateFormat

При форматировании даты и времени в соответствии с региональными стандартами следует иметь в виду четыре особенности:

- названия месяцев и дней недели должны быть представлены на местном языке;
- последовательность указания года, месяца и числа различаются для разных стран и регионов;
- для отображения дат можно использовать календарь, отличный от григорианского;
- следует учитывать часовые пояса.

Для учета перечисленных возможносте в Java имеется класс **DateFormat**, который используется почти также, как и класс *NumberFormat*. В первую очередь следует сформировать объект регионального стандарта. Для получения массива региональных стандартов, поддерживающих формат даты, можно использовать предлагаемый по умолчанию статический метод *getAvailableLocales ()*.

```
fmt = DateFormat.getDateInstance     (dateStyle, loc);
fmt = DateFormat.getTimeInstance     (timeStyle, loc);
fmt = DateFormat.getDateTimeInstance (dateStyle,
                                      timeStyle, loc);
```

```
DateFormat.DEFAULT;
DateFormat.FULL   ; // Wednesday, Septemer 15 2004, 8:15:03 pm
                    // для регионального стандарта США
DateFormat.LONG   ; // Septemer 15, 2004 8:15:03 pm
                    // для регионального стандарта США
DateFormat.MEDIUM ; // Sep 15, 2004 8:15:03 pm
                    // для регионального стандарта США
DateFormat.SHORT  ; // 9/15/04 8:15 pm
                    // для регионального стандарта США
```

Определение файла ресурсов ResourceBundle

Для локализации приложений создаются так называемые *пакеты ресурсов* (**resource bundle**). Каждый пакет представляет собой файл свойств или класс, который описывает элементы, специфические для конкретного регионального стандарта (например, сообщения, надписи и т.д.). В каждый пакет помещаются ресурсы для всех региональных стандартов, поддержка которых предполагается в программе.

Для именования пакетов ресурсов используются специальные соглашения. Например, ресурсы, специфические для Германии, помещаются в файл с именем *имяПакета_de_DE*, а ресурсы, общие для стран, в которых используется немецкий язык, размещаются в классе *имяПакета_de*. Общие правила таковы : ресурсы для конкретной страны именуются по принципу:

имяПакета_язык_СТРАНА

Имя файла ресурсов для конкретного языка формируется так :

имяПакета_язык

Ресурсы, применяемые по умолчанию, помещаются в файл, имя которого не содержит суффикса. Для загрузки пакета ресурсов используется метод **getBundle()**.

Методологии разработки ПО

Разработка ПО – это сложный бизнес-процесс. А значит, ИТ нужно разговаривать на языке оптимизации, планирования и расчета.

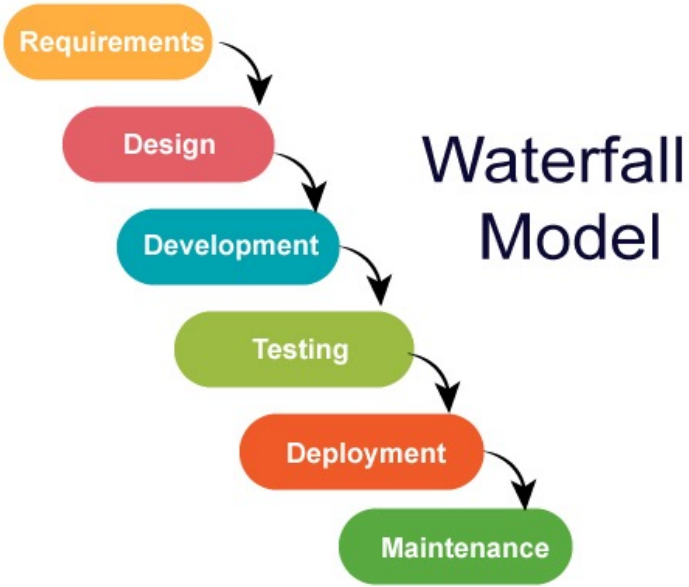
Важное уточнение для начала: есть отдельно ***модели*** разработки ПО и отдельно — ***методологии*** этой самой разработки. Модели предсказывают будущее поведение системы. Методологии нужны, чтобы система работала, причем так, как нужно.

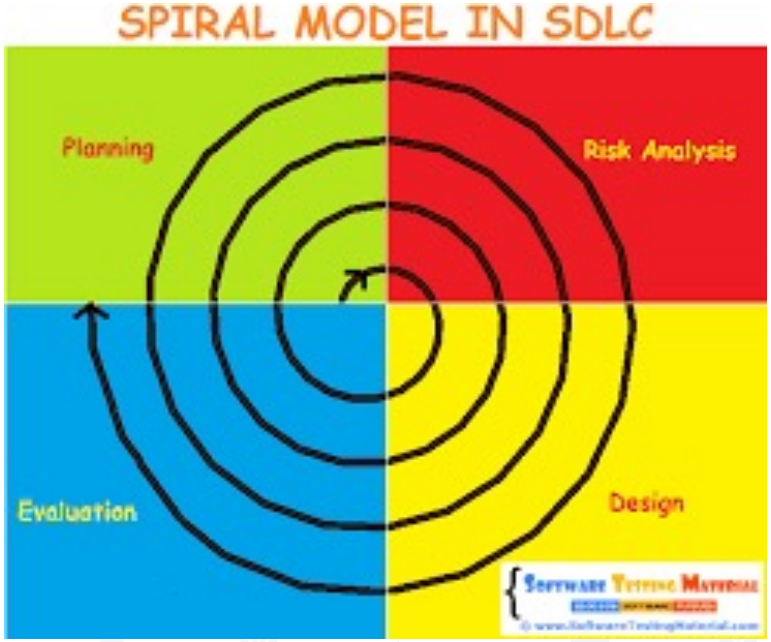
Путать модели и методологии разработки ПО — святое дело каждого ИТ-новичка, поэтому грубой ошибкой это не считается.

И все же, модели – это ***классическая каскадная*** ***Waterfall***, с ее линейностью, четкой постановкой целей для каждого этапа и строгим контролем за сроками.

Модели — это ***Spiral***, с ее фокусировкой на раннем выявлении и уменьшении проектных рисков. Разработка по Spiral начинается в небольшом масштабе, сначала решаются локальные задачи, а потом - более комплексные.

И наконец, модели — это ***IID***, в которой жизненный цикл проекта разбивается на последовательность итераций, каждая из которых напоминает «мини-проект». В общем, **модель** — это то, что **описывает процесс разработки ПО**.


Requirements
Design
Development
Testing
Deployment
Maintenance
Waterfall Model


SPIRAL MODEL IN SDLC
Planning
Risk Analysis
Evaluation
Design

А вот **методологии — это системы контроля, оценки и мониторинга работы над поставленными задачами.** Методологии — это кнут и пряник современного разлива, которые нужны для контроля каждого звена процесса разработки.

Методология «Scrum»

Scrum — **гибкий метод управления проектами**. В его основе лежат «спринты» — короткие итерации, строго ограниченные по времени (как правило, 2-4 недели). Продолжительность совещаний при этом сводится к минимуму, но увеличивается их частота. Каждый спринт состоит из списка задач до конца итерации, и у каждой из них свой “вес”. Во время совещаний команда обсуждает, кто что сделал, что собирается сделать и какие есть проблемы. Для планирования в Scrum используют журнал спринтов. В таком подходе в команде часто появляется **Scrum-мастер**, который налаживает непрерывную работу всей команды, создавая для нее комфортные условия. Также на проекте появляется роль **Product Owner** — руководителя разработки, человека, который следит за продуктом и выступает главным звеном между запросом клиента и результатом команды.

Плюсы:

- быстрый запуск проекта с минимально возможным бюджетом;
- ежедневный контроль над ходом работ, частые демонстрации проекта;
- возможность вносить правки по ходу реализации проекта.

Минусы:

- сложности при заключении договоров из-за отсутствия фиксированного бюджета;
- не работает при низкой квалификации команды, заниженных сроках работ или бюджете;
- из-за возможности постоянно вносить изменения между спринтами может создавать путаницу.

Методология «Kanban»

Важнейшая особенность Kanban — **визуализация жизненного цикла проекта**. Создаются колонки выполнения задач, которые сдаются индивидуально. Колонки обозначены маркерами типа: **To do, In progress, Code review, In testing, Done** (название колонок, конечно же, может меняться). Цель каждого участника команды — уменьшать количество задач в первой колонке. Подход Kanban наглядный и помогает понять, где возникла проблема. Структуру по Kanban не определяют окончательно и бесповоротно: в зависимости от специфики проекта можно добавить импровизированные колонки. Например, некоторые команды используют систему, в которой нужно определить критерии готовности задачи перед ее выполнением. Тогда добавляют две колонки — specify (уточнить параметры) и execute (взяться за работу).

Плюсы:

- гибкость планирования. Команда концентрируется только на текущей работе, приоритет задачи также определен;
- наглядность. Когда у всех исполнителей есть доступ к данным, глобальную проблематику легче заметить;
- высокая вовлеченность в процесс разработки. Визуализация процессов повышает самоорганизацию и самоконтроль.

Минусы:

- не работает с командами численностью более пяти человек;
- не предназначена для долгосрочного планирования;
- не подходит для работы в команде без мотивации. В Kanban не существует сроков, установленных по каждой задаче, не предусматривает методология и штрафов за просрочку.

Методология «RUP»

Методология RUP использует итеративную модель разработки. В конце каждой итерации (которая занимает от 2 до 6 недель) команда должна достичь запланированных целей и получить временную, но работающую версию проекта. RUP предполагает **разделение проекта на четыре фазы**, в каждой из которых ведется работа над новым поколением продукта: фазу начала проекта, уточнение, построение и внедрение. По окончании фазы вводится маркер завершения этапа (Project Milestone). Project Milestone можно считать моментом, когда команда дает оценку достигнутым результатам. В итоге методология подразумевает, что на первом этапе выпускаются основные функции, а на последующих фазах добавляются дополнения.

Плюсы:

- позволяет справляться с меняющимися задачами, исходящими как от клиента, так и возникающими в ходе работы;
- обеспечивает постоянное улучшение продукта. Во время итераций можно скрупулезно оценить проект;
- позволяет определять и устранять риски на ранних стадиях работы, а также эффективно контролировать качество разработки.

Минусы:

- довольно сложный метод, который трудно внедрить при небольшой команде или компании;
- завязанность на способности экспертов ставить задачи;
- нуждается в излишнем документировании требований.