Вопросы к защите 5 лабы ПИП

Интернационализация и локализация

Локализация – перевод, адаптация и техподдержка ПО на языке какой-либо страны, добавление специальных функций для использования ПО в каком-то определенном регионе.

Интернационализация – адаптация продукта для потенциального использования практически в любом месте.

Интернационализация производится на начальных этапах разработки, в то время как локализация – для каждого целевого языка в конце разработки.

Средства интернационализации:

- Упрощение перевода (строки хранятся в отдельных файлах)
- Поддержка стандартов (использование стандартных функций там, где это возможно)

Способы хранения локализованных данных. Классы ResourceBundle, PropertyResourceBundle, ListResourceBundle.

Классы, предназначенные для локализации, призваны обеспечить удобство адаптации ПО под определенный регион.

Класс ResourceBundle – абстрактный класс, описывающий наиболее общее поведение.

Наследники класса ListResourceBundle, который, кстати говоря, тоже абстрактный, должны переопределять метод getContents(), возвращающий двумерный массив объектов, содержащий пары значений, первое из которых выступает в роли ключа, второе – в роли локализованного объекта. Ключи, само собой, чувствительны к регистру.

Класс PropertyResourceBundle (уже не абстрактный) хранит локализованные данные в том же самом формате, однако, в отличие от двух предыдущих классов, для использования PropertyResourceBundle необходимо не унаследоваться от абстрактного класса, а предоставить объекту PropertyResourceBundle файл с локализованным данными (property-file).

PropertyResourceBundle имеет два конструктора – один принимает объект типа InputStream (поток байтов), который работает только с кодировкой ISO-8859-1. Второй же конструктор принимает объект типа Reader и может работать с любой кодировкой. Есть еще один вариант – использовать метод getBundle(), в который нужно передать BaseName (базовое имя файла с локализацией, то есть та часть имени файла с расширением .properties, которая от файла к файлу остается постоянной, например, MyResources) и local (та часть имени, которая меняется от файла к файлу, например, ru_RU). Еще можно передавать в качестве третьего параметра объект ClassLoader.

Форматирование числовых данных. Классы NumberFormat, DecimalFormat, DecimalFormatSymbols.

Классы *Format* предназначены для форматирования числовых данных для различных регионов (связано с локализацией). Пример использования NumberFormat:

String s = NumberFormat.getInstance(Loacalization.FR).format(myNumber);

Также есть методы:

- getIntegerInstance – для целых чисел (666.1313 отобразится как 666)
- getCurrencyInstance – для валюты (666.1313 отобразится как 666.13 руб. или $666.13)
- getPercentInstance – для процентов (666.1313 отобразится как 66.613%)

Можно контролировать количество знаков после запятой:

```
NumberFormat numberFormat = NumberFormat.getCurrencyInstance(Locale.CHINESE);
numberFormat.setMinimumFractionDigits(14);
System.out.println(numberFormat.format(myNumber));
```

Или перед запятой:

```
NumberFormat numberFormat = NumberFormat.getCurrencyInstance(Locale.CANADA);
numberFormat.setMinimumFractionDigits(14);
numberFormat.setMinimumIntegerDigits(12);
System.out.println(numberFormat.format(myNumber));
```

DecimalFormat используется для парсинга чисел в форматы, зависящие от конкретного региона. DecimalFormat позволяет определять паттерны для форматирования чисел. Примеры использования:

```
DecimalFormat decimalFormat = new DecimalFormat("####,###");
System.out.println(decimalFormat.format(myNumber));
//output : 666

decimalFormat.applyPattern("вот твое число : 000.00000");
System.out.println(decimalFormat.format(myNumber));
//output : вот твое число : 666,13130

decimalFormat.applyPattern("0000.0000");
System.out.println(decimalFormat.format(myNumber));
//output : 0666,1313

NumberFormat nf = NumberFormat.getInstance(Locale.ENGLISH);
DecimalFormat df = (DecimalFormat)nf;
df.applyPattern("####,##.#");
System.out.println(df.format(myNumber));
//In the England symbol '.' - separator of integer and float parts
//output : 6,66.1

nf = NumberFormat.getInstance(Locale.GERMAN);
df = (DecimalFormat)nf;
df.applyPattern("####,##.#");
System.out.println(df.format(myNumber));
//In the German symbol ',' - separator of integer and float parts
//output : 6.66,1
```

У DecimalFormat есть методы applyPattern и applyLocalizedPattern. Эти два метода отличаются тем, что applyPattern интерпретирует символы в паттерне как стандартные символы для составления паттерна, а applyLocalizedPattern может интерпретировать эти символы по-разному в зависимости от локализации. Пример:

```
decimalFormat.applyPattern("###,##.#");
System.out.println(decimalFormat.format(myNumber));
//',' - separator in integer part (in our localization it is ' ')
//'.' - separator between integer and float part (in our localization it is ',')
//output : 6 66,1

decimalFormat.applyLocalizedPattern("###,##.#");
System.out.println(decimalFormat.format(myNumber));
//',' - in our localization it is separator of integer and float part
//'.' - in our localization it is something strange
//output : 666,131.
```

DecimalFormatSymbols позволяет нам поменять стандартные символы, отображающиеся при форматировании числа (например, сделать букву а разделителем целой и дробной части, а букву б – разделителем групп чисел в целой части).

```
DecimalFormatSymbols decimalFormatSymbols = new DecimalFormatSymbols();
decimalFormatSymbols.setCurrencySymbol(" бабла");
decimalFormatSymbols.setDecimalSeparator('a');
```

```
decimalFormatSymbols.setGroupingSeparator('б');

df = new DecimalFormat("##,##.##",decimalFormatSymbols);

System.out.println(df.format(myNumber));
//output : 6б66а13

nf = NumberFormat.getCurrencyInstance();
((DecimalFormat) nf).setDecimalFormatSymbols(decimalFormatSymbols);
System.out.println(nf.format(myNumber));
//output : 666,13 бабла
```

Форматирование даты и времени. Классы DateFormat, SimpleDateFormat, DateFormatSymbols.

DateFormat предназначен для преобразования объектов типа Date в строку (format) и наоборот (parse). Это избавляет от необходимости ручного транслейта, к примеру, дней недели или названий месяцев.

```
DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.FULL, Locale.GERMAN);
System.out.println(dateFormat.format(date));
//output : Sonntag, 20. November 2016

dateFormat = DateFormat.getDateInstance(DateFormat.LONG, Locale.CHINA);
System.out.println(dateFormat.format(date));
//output : 2016年11月20日

dateFormat.setCalendar(Calendar.getInstance(Locale.TAIWAN));
System.out.println(dateFormat.format(date));
//output : 2016年11月20日
```

SimpleDateFormat позволяет использовать паттерны

```
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("Эра : GGGG\nГод : YYYY\nМесяц
: MMMM\nДень : dd\n"+
        "День от начала года : DD\nЧас : HH\nМинута : mm\nСекунда : ss\nМиллисекунда :
SS"+
        "\nНеделя в году : ww\nНеделя в месяце : WW");
System.out.println(simpleDateFormat.format(date));
```

```
Вывод:
Эра : н.э.
Год : 2016
Месяц : ноября
День : 20
День от начала года : 325
Час : 15
Минута : 10
Секунда : 56
Миллисекунда : 539
Неделя в году : 47
Неделя в месяце : 03
```

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
Y	Week year	Year	2009; 09
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day name in week	Text	Tuesday; Tue
u	Day number of week (1 = Monday, ..., 7 = Sunday)	Number	1
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
X	Time zone	ISO 8601 time zone	-08; -0800; -08:00

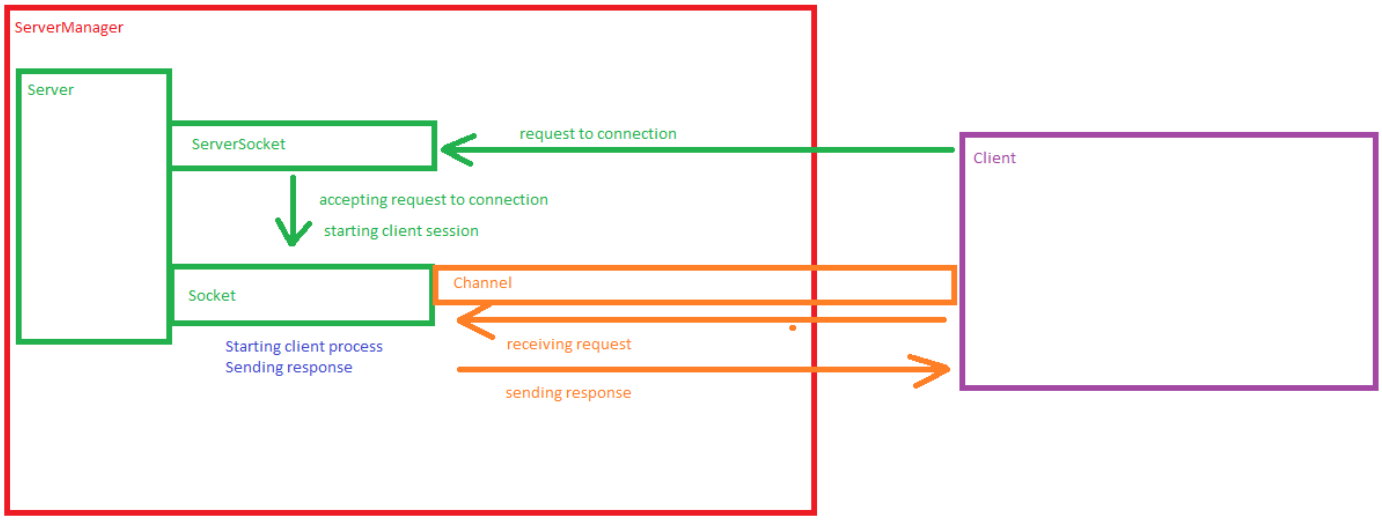
Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, ''yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"	2001-07-04T12:08:56.235-07:00
"YYYY-'W'ww-u"	2001-W27-3

DateFormatSymbols

```
DateFormatSymbols dateFormatSymbols = new DateFormatSymbols();
dateFormatSymbols.setWeekdays(new String[]{"ПН","ВТ","СР","ЧТ","ПТ","СБ","ВС"});
simpleDateFormat = new SimpleDateFormat("EEEE-YYYY-MMMM",dateFormatSymbols);
System.out.println(simpleDateFormat.format(date));
```

Вывод : ВТ-2016-ноября

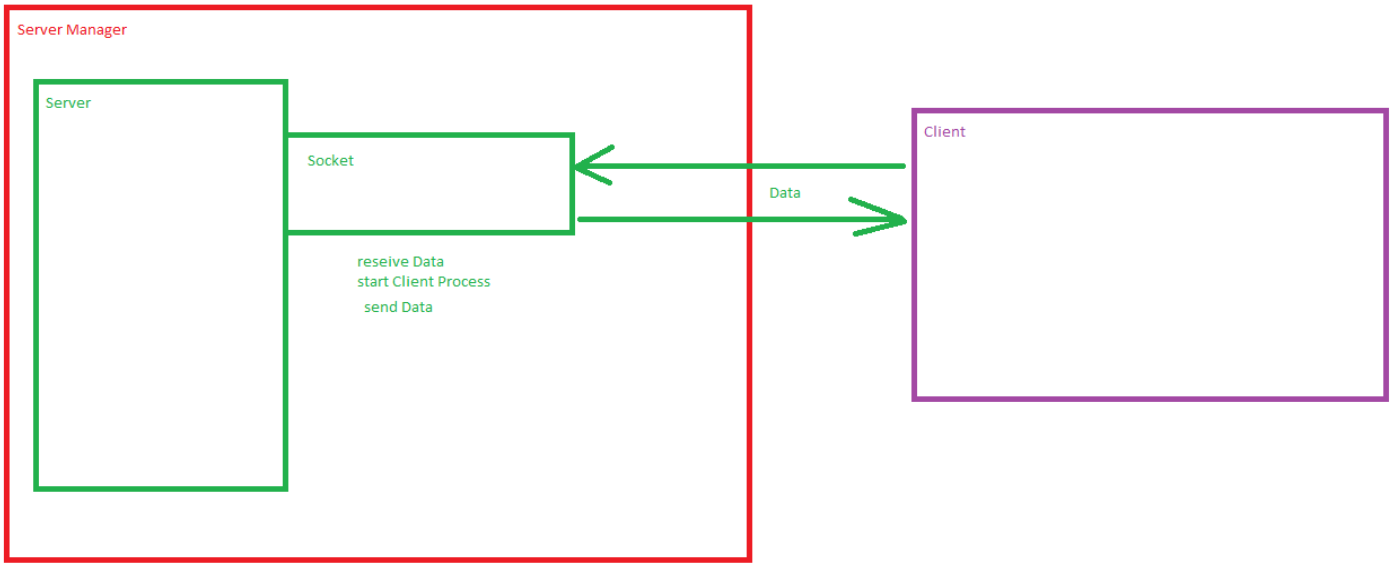
Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.



Для создания клиент – серверной архитектуры на протоколе TCP нужно создать несколько классов:

- ServerManager – управляет сервером, запускает и останавливает его

- Server – представляет собой приложение-сервер
- ClientSession – представляет поток исполнения, обрабатывающий последовательные запросы от одного клиента, пока клиент подключен к серверу
- Channel – абстракция канала передачи данных, тут осуществляется сериализация и десериализация объектов для передачи информации по сети
- ClientProcess – представляет поток исполнения, обрабаотывающий отдельный запрос от клиента
- Request – представляет объекты – запросы
- Response – представляет объекты – ответы на запросы
- Client – объект, отправляющий серверу запросы на обслуживание



Организовать работу по протоколу UDP несколько проще, нам нужны классы:

- ServerManager – будет управлять сервером, запускать и останавливать его
- Server – приложение, к которому производится подключение для запроса на обработку данных
- ClientProcess – запускается как только данные приняты и можно начинать их обработку
- Client – тот, кто будет отправлять запросы и принимать ответы
- DataConverter – класс, который будет преобразовывать данные для передачи в DatagramPacket

Про основные протоколы, их сходства и отличия можно почитать здесь

Протокол TCP. Классы Socket и ServerSocket.

Класс Socket – структура данных, с помощью которой упрощается передача данных по сети, в частности, мы можем производить идентификацию приложений с помощью Socket, но не только это. Класс Socket представляет собой что-то вроде «разъема», если проводить аналогию, к которому можно подключиться и через который можно обмениваться информацией. Разъем однозначно идентифицируется с помощью адреса компьютера в сети и порта (адреса процесса на этом компьютере).

Класс ServerSocket предназначен для того, чтобы порождать объекты Socket как только обнаружится новый запрос на подключение. Это значит, что, когда клиент хочет приконнектиться к серверу, то он использует SocketAddress объекта ServerSocket, но реально, когда происходит подключение, на стороне сервера срабатывает метод ServerSocket.accept() и происходит создание нового объекта Socket, через который клиент и сервер организовывают взаимодействие.

Про протокол TCP много чего написано и в интернете, можно немного узнать о нем здесь.

Протокол UDP. Классы DatagramSocket и DatagramPacket.

Протокол UDP, в отличие от TCP, не устанавливает соединение предварительно, а сразу «кидает» данные в сеть.

Естественно, должен быть какой-то способ для указания получателя данных. Чтобы это было проще сделать, придумали тип данных DatagramPacket, экземпляры которого хранят в себе байтовый массив данных, длину этого массива, а также адрес компьютера-получателя и порт на этом компьютере.

Для отправки и получения данных и на клиентской, и на серверной стороне используют объект типа DatagramSocket, предназначение которого почти то же самое, что и у класса Socket – однозначно идентифицировать «сетевые узлы».

Более подробно о протоколе UDP можно узнать на википедии или здесь.

Передача данных по сети. Сериализация объектов, интерфейс Serializable.

При передаче данных с помощью протокола TCP используется объекты типа OutputStream и InputStream. Процесс преобразования объекта типа Request в форму, пригодную для записи в OutputStream (то есть в форму последовательности, «серии» байт) называется сериализацией, обратный процесс восстановления из InputStream в объект, пригодный для работы называется десериализацией. Типы данных, объекты которых могут сериализовываться или десериализовываться, должны реализовывать интерфейс Serializable, который не обязывает программиста к реализации каких-то специфических методов, а является чем-то наподобие метки или тега.

В принципе, при использовании протокола UDP также есть необходимость преобразования объекта в массив байтов, и если мы не имеем дело с числовыми данными, конверт которых можно провести штатными средствами, то также имеет место сериализация и десериализация.

Механизм рефлексии (reflection) в Java. Класс Class.

Механизм рефлексии в джава – механизм, который позволяет программисту получать и использовать объекты, представляющие собой на высоком уровне абстракции не какие-то отдельные сущности, а другие классы, методы и прочее. Например, при написании юнит-тестов мы иногда сталкиваемся с необходимостью использования подобного вида записи:

```
@Test(expected = ArithmeticException.class)
```

Тут мы делаем не что иное, как получаем объект, представляющий собой класс ArithmeticException. Естественно, у объектов типов, предусматриваемых механизмом рефлексии (а именно, Method, Class, Properties) есть много полезных методов – к примеру, можно без труда получить список конструкторов какого-то класса, значение определенного свойства, создать экземпляр и так далее. Более подробно механизм рефлексии в джава описан на сайте оракл.