# Correction Localization - 0

A - Arrays.stream(loc).filter(x -> x.getLanguage().equals(“fr”)).forEach(System.out::println);

D - Arrays.stream(loc).filter(x -> x.toString().startswWith(“fr”)).forEach(System.out::println);

‘public static Locale[] getAvailableLocales() {…}’ returns the Locale [] containing all the available locales supported by the JVM.

Language codes are stored in lower case, so syntax comparing upper case codes (‘FR’) is ruled out.

You are left with 3 options [working with lower case language code (‘fr’)]. ‘x’ in the lambda expression is of Locale type and Locale class doesn’t have method startsWith, so ‘x -> x.startsWith(“fr”)’ causes compilation failure.

‘x -> x.getLanguage().equals(“fr”)’ and ‘x -> x.toString().startsWith(“fr”)’ correctly filters out expected Locale objects.

# Correction Localization - 1

B - French/Canada

ResourceBundle.getBundle(“ResourceBundle”, loc); => Base resource bundle file name should be ‘ResourceBundle’.

Default Locale is: fr_CA and passed Locale to getBundle method is: en_IN

The search order for matching resource bundle is:

ResourceBundle_en_IN.properties [1st: Complete en_IN].

ResourceBundle_en.properties [2nd: Only language en].

ResourceBundle_fr_CA.properties [3rd: Complete default Locale fr_CA].

ResourceBundle_fr.properties [4th: Language of default Locale fr].

ResourceBundle.properties [5th: ResourceBundle’s name without language or country].

Out of the given resource bundles, ‘ResourceBundle.properties’ matches.

This resource bundle has key ‘locale’ and value ‘French/Canada’.

rb.getObject(“locale”) prints ‘French/Canada’ on to the console.

# Correction Localization - 2

C -

```
key1=ONE
key2=TWO
key3=THREE
```

If all the files have same keys and locale is en_US, then key-value pair of best match (‘RB_en_US.properties’) file will be considered.

If the files have different keys and locale is en_US, then key-value pair is searched in following order:

1. RB_en_US.properties [parent = ‘RB_en.properties’]
2. RB_en.properties [parent = ‘RB.properties’]
3. RB.properties [parent = null]

Hence bundle.getKeys(); returns an Enumeration instance containing (“key1”, “key2”) from ‘RB_en_US.properties’ file and (“key3”) from ‘RB_en.properties’ file.

I mentioned parent value abvoe because there is relationship among above 3 files. RB.properties is the ultimate parent, its child is ‘RB_en.properties’ and ‘RB_en_US.properties is the child of ’RB_en.properties’.

So if I write below codes:

bundle.getString(“key1”); value will be taken from ‘RB_en_US.properties’ file, so it returns ONE.

bundle.getString(“key2”); value will be taken from ‘RB_en_US.properties’ file, so it returns ONE.

msgs.getString(“key3”); this key is not available in ‘RB_en_US.properties’ file and hence it will be searched in its parent ‘RB_en.properties’. And because “key3” key is available in ‘RB_en.properties’ file, hence it returns THREE.

You can iterate over Enumeration using hasMoreElements() and nextElement() methods.

Iteration logic prints key-value pair in the order of Enumeration elements (“key1”, “key2”, “key3”). Hence output will be:

```
key1=ONE
```

```
key2=TWO
key3=THREE
```

If you read the javadoc of getKeys(), getString(String) or getObject(String) methods, then you will find out that these methods work with current resource bundle and its parent.

# Correction Localization - 3

B -

```
Good Morning!
Good Evening!
Good Day!
null
```

java.util.Properties extends Hashtable<Object, Object>, which is a map to store key value pairs.

load method accepts InputStream object and loads the key value pairs (present in InputStream object) to internal map of Properties class.

There are 2 overloaded versions of getProperty() method:

1. String getProperty(String key) => Returns value of the specified key and null if value is not found.
2. String getProperty(String key, String defaultValue) => Returns value of the specified key and defaultValue if value is not found.

‘key1’ and ‘key2’ are present in Properties file, so their corresponding values are returned.

As ‘key3’ is not present in the Properties file, hence ‘prop.getProperty(“key3”, “Good Day!”)’ returns the defaultValue, “Good Day!” and as ‘key4’ is not present as well, hence ‘prop.getProperty(“key4”)’ returns null.

# Correction Localization - 4

E -

```
temp: UNKNOWN
temp_UNKNOWN
```

Locale class has overloaded constructors:

Locale(String language)

Locale(String language, String country)

Locale(String language, String country, String variant)

For the exam, you should know that Locale instance can be created by passing incorrect country and language and in fact getLanguage(), getCountry() and toString() method prints the passed strings.

NOTE: It may cause problem later on, when you try to retrieve resource bundle.

# Correction Localization - 5

A - Runtime Exception

ResourceBundle.getBundle(“MyResourceBundle”, loc); => Base resource bundle file name should be ‘MyResourceBundle’.

Default Locale is: fr_CA and passed Locale to getBundle method is: en_IN

The search order for matching resource bundle is:

MyResourceBundle_en_IN.properties [1st: Complete en_IN].

MyResourceBundle_en.properties [2nd: Only language en].

MyResourceBundle_fr_CA.properties [3rd: Complete default Locale fr_CA].

MyResourceBundle_fr.properties [4th: Language of default Locale fr].

MyResourceBundle.properties [5th: ResourceBundle’s name without language or country].

None of the properties files match with above search list, hence MissingResourceException is thrown at runtime.

# Correction Localization - 6

B - Code compiles and executes successfully and prints it_IT on to the console.

setDefault(Locale) is a static method defined inside Locale class but static method can be invoked using instance reference as well.

Compiler changes ‘loc.setDefault(loc);’ to ‘Locale.setDefault(loc);’.

Line 8 doesn’t cause any compilation error and on execution it sets the default locale to it_IT.

Hence, Locale.getDefault() will always return it_IT.

# Correction Localization - 7

A - Line n4 causes compilation failure

parse(…) method defined in new Date/Time API such as LocalDate, LocalTime, LocalDateTime and DateTimeFormatter can throw java.time.DateTimeException (which is a RuntimeException) but parse(…) method defined in NumberFormat class declares to throw java.text.ParseException (which is a checked exception) and hence needs to follow declare or handle rule.

As main(…) method doesn’t provide try-catch block around Lind n4 and also it doesn’t declares to throw ParseException, so Line n4 causes compilation failure.

# Correction Localization - 8

D - Locale.getDefault();

Locale.getDefault(); returns the default Locale for the currently running instance of JVM and getDefault() method is from Locale class and not from System class.

# Correction Localization - 9

A - Locale loc4 = new Locale(“”,””);

B - Locale loc3 = new Locale(“”);

D - Locale loc5 = new Locale(“”,””,””);

Question only talks about successful creation of Locale instance.

Locale class has 3 public overloaded constructors:

Locale(String language)

Locale(String language, String country)

Locale(String language, String country, String variant)

But constructor with no-argument and constructor that accepts another Locale instance are not available. Hence ‘new Locale();’ and ‘new Locale(loc1);’ causes compilation error.

Passing blank string is OK and Locale objects are successfully created.

NOTE: If you pass null as argument to any of the overloaded constructors, then NullPointerException will be thrown and Locale objects will not be successfully created in that case.

# Correction Localization - 10

C - Given code compiles successfully

getDisplayCountry() and getDisplayLanguage() methods are overloaded to accept Locale type.

Given Test class compiles successfully.

# Correction Localization - 11

E - Compilation error

There are no setCountry(…) and setLanguage(…) methods defined in Locale class.

# Correction Localization - 12

B - 1001

The search order for matching resource bundle is:

com.training.ocp.MyResourceBundle_en_US [1st: Complete, en_US].

com.training.ocp.MyResourceBundle_en [2nd: Only language, en].

com.training.ocp.MyResourceBundle_fr_IT [3rd: Complete Default Locale, fr_IT].

com.training.ocp.MyResourceBundle_fr [4th: Language of Default Locale, fr].

com.training.ocp.MyResourceBundle [5th: ResourceBundle's name without language or country].

If search reaches the 5th step and no matching resource bundle is found, then MissingResourceException is thrown at runtime.

In 4th step, matching resource bundle is found and hence '1001' is printed on to the console.

# Correction Localization - 13

B -

```
k3=EN3
k4=EN4
k1=1
k2=2
```

Locale.US represents a Locale instance for en_US. As locale is en_US, hence given files will be searched in below order:

ResourceBundle_en_US.properties (Not available)

ResourceBundle_en.properties (Available and works even in the file name “EN” is in upper-case, but convention is to use lowercase of language code)

ResourceBundle.properties (Available)

Note that ‘ResourceBundle_US.properties’ will not be considered.

Hence bundle.getKeys(); returns an Enumeration instance containing (k3,k4) from ‘ResourceBundle_EN.properties’ file and (k1,k2) from ‘ResouceBundle.properties’ file.

You can iterate over Enumeration using hasMoreElements() and nextElement() methods.

Iteration logic prints key-value pair in the order of Enumeration elements (k3, k4, k1, k2). Hence the output will be:

```
k3=EN3
k4=EN4
k1=1
k2=2
```

# Correction Localization - 14

C - Locale l1 = Locale.US;

D - Locale l5 = new Locale(“en”, “US”);

Locale.US; => Locale.US represents a Locale instance for en_US.

new Locale(Locale.US); => There is no Locale constructor which accepts a Locale object.

Locale.getInstance(“us”); => There is no getInstance method, which accepts single String argument. Famous getInstance method has signature: ‘getInstance(String language, String country, String variant)’.

Locale.getDefault(); => It will not always return Locale for en_US, it depends on the default locale of JVM.

new Locale(“en”, “US”); => This definitely creates a Locale instance for en_US.

# Correction Localization - 15

B - Amount $10.00 is in USD

As Locale instance is for en_US, hence ‘NumberFormat.getCurrencyInstance(loc);’ returns NumberFormat instance for US.

Currency code for US is USD and it is represented as $.

nf.getCurrency() returns the Currency object with currency Code “USD” and nf.format(10) returns $10.00

# Correction Localization - 16

D -

```
country=Sri Lanka
continent=Asia
```

Format of resource bundle properties file is:

key1=value1 key2=value2

There must be a newline between 2 pairs of key and value. Colon(:) and Semicolon(;) are not used as separators.

Even if value has space in between, double quotes are not used.

NOTE: Generally as a good convention spaces in keys are not OK but if you have to have the space, then escape it properly.

country name=Sri Lanka => NOT OK.

country name=Sri Lanka => OK.

# Correction Localization - 17

C - SURPRISE!

The search order for matching resource bundle is:

com.training.ocp.MyResourceBundle_en_US [1st: Complete, en_US].

com.training.ocp.MyResourceBundle_en [2nd: Only language, en].

com.training.ocp.MyResourceBundle_fr_IT [3rd: Complete Default Locale, fr_IT].

com.training.ocp.MyResourceBundle_fr [4th: Language of Default Locale, fr].

com.training.ocp.MyResourceBundle [5th: ResourceBundle's name without language or country].

If search reaches the 5th step and no matching resource bundle is found, then MissingResourceException is thrown at runtime.

In 5th step, matching resource bundle is found and hence 'SURPRISE!' is printed on to the console.

# Correction Localization - 18

B -

```
true
false
```

Builder is a static nested class added in Locale class in JDK 7.

NOTE: Builder's region is same as Locale's country.

l1 refers to Locale object for 'en_US'. l2 also refers to Locale object for 'en_US' but l3 doesn't refer to Locale object for 'en_US'. It just refers to Locale object for 'en'.

In fact, l3.getCountry() and l3.getDisplayCountry() return blank string.

l1 and l2 are same but l3 is different.

# Correction Localization - 19

E - No text is displayed in the output

Locale.ENGLISH is equivalent to new Locale(“en”, ““);

So, language is ‘en’ and country is blank.

loc.getDisplayCountry() returns blank string, hence no text is displayed in the output.

# Correction Localization - 20

B - ResourceBundle.properties

Default Locale is: fr_CA and passed Locale to getBundle method is: en_IN

The search order for matching resource bundle is:

ResourceBundle_en_IN.properties [1st: Complete en_IN].

ResourceBundle_en.properties [2nd: Only language en].

ResourceBundle_fr_CA.properties [3rd: Complete default Locale fr_CA].

ResourceBundle_fr.properties [4th: Language of default Locale fr].

ResourceBundle.properties [5th: ResourceBundle's name without language or country].

Out of the given resource bundles, 'ResourceBundle.properties' matches.

# Correction Localization - 21

B - getCurrencyInstance()

D - getCurrencyInstance(java.util.Locale.US)

As expected output is: $5.00, which means formatter must be for the currency and not the number.

NumberFormat.getInstance() and NumberFormat.getInstance(Locale) return the formatter for the number and hence will display 5 on to the console.

NumberFormat.getCurrencyInstance() returns the currency formatter for default Locale which is en_US, hence format(5) will display $5.00 on to the console.

NumberFormat.getCurrencyInstance(java.util.Locale.US) returns the currency formatter for the specified Locale, which is again en_US, hence format(5) will display $5.00 on to the console.

# Correction Localization - 22

D - abstract protected Object[][] getContents();

ListResourceBundle has one abstract method: ‘abstract protected Object[][] getContents();’.

All the concrete sub classes of ListResourceBundle must override this method.

# Correction Localization - 23

B - Locale [] loc = Locale.getAvailableLocales();

C - Object [] locale = Locale.getAvailableLocales();

‘public static Locale[] getAvailableLocales() {…}’ returns the Locale [] containing all the available locales supported by the JVM. Hence, ‘Locale [] loc = Locale.getAvailableLocales();’ will compile successfully.

As Object is the ultimate base class, hence ‘Object [] locale = Locale.getAvailableLocales();’ also works.

NOTE: Object [] is not the parent of Locale [] but array allows above syntax because it has ArrayStoreException.