INTRODUCING "JAVA 8"

Language eehancements, lambda etc.

Java 8 Language features

- Lambda Expressions
- Annotations Enhancement
- A new Date/Time API

Other....

- Compact Profiles
- Internationalization enhancements

Lambda Expressions

- Many languages (usually weakly typed) let you pass functions around.
- Functional approach proven concise, useful and parallelizable
- eg. [Javascript code]

```
var t1=["one","two","three","four"];
t1.sort(function (s1,s2){
         returns (s1.length - s2.length);});
```

Concise and Expressive

- Old

```
 button.addActionListener( new ActionListener()
  { @Override
       public void actionPerformed(ActionEvent e) {
             blah(); }
  });
```

- New

```
button.addActionListener(e -> blah());
```

Most basic form

- You write what looks like a function
  - Arrays.sort(testString, (s1,s2) -> s1.length()-s2.length());
  - someButton.addActionListener(e -> handleClick());
- You get an instance of class that implements the interface that was expected in that place.
  - expected type must be an interface that has exactly one (abstract) method.
  - Called Functional Interface or Single Abstract Method interface.

Where can be used?

- Find any variable or parameter that expects an interface that has one method
  - eg. Runnable, ActionListener, Comparator, Comparable etc.
- Code that uses Interface is the same
  - code that uses the interface must still know the real method name.
- Code that calls interface can supply lambda.
  - Thread t = new Thread(()->System.out.println(“Hello”));

Syntax

- Replace this:

```
new SomeInterface() {
  @Override public SomeType someMethod(args) {
      body
  } }
```

- With this:

```
(args) -> { body }
```

Example

- **Old style:**

```
Arrays.sort(testStrings, new Comparator<String>() {
  @Override public int compare(String s1, String s2)
  { return(s1.length() -s2.length()); } });
```

- **New Style:**

```
Arrays.sort(testStrings,
  (String s1, String s2) -> { return(s1.length() –
  s2.length()); });
```

Annotations Enhancements

- Annotations on Java Types
- Repeating Annotations
- Method parameter reflection

Annotations

- *Annotations*, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.
- Annotations have a number of uses, among them:
  - **Compile-time and deployment-time processing**

    Software tools can process annotation information to generate code, XML files, and so forth.
  - **Runtime processing**

    Some annotations are available to be examined at runtime.
  - **Information for the compiler**

    Annotations can be used by the compiler to detect errors or suppress warnings.

Annotation on Types

- Before the Java SE 8 release, annotations could only be applied to declarations. As of the Java SE 8 release, annotations can also be applied to any *type use*.
- A few examples of where types are used are class instance creation expressions (new), casts, implements clauses, and throws clauses.

Annotations on Types

- Class instance creation expression:
```
new @Interned MyObject();
```
- Type cast:
```
myString = (@NonNull String) str;
```
- implements clause:
```
class UnmodifiableList<T> implements @Readonly
List<@Readonly T> { ... }
```
- Thrown exception declaration:
```
void monitorTemperature() throws @Critical
TemperatureException { ... }
```

New Date / Time API

- The Date-Time package, java.time, introduced in the Java SE 8 release, provides a comprehensive model for date and time and was developed under JSR 310: Date and Time API.
- Supports Standard time cocepts
  - date, time, instant and timezone
- provides limited set of calendar systems and extensible to others

Date Time API

- The core classes in the Date-Time API have names such as LocalDateTime, ZonedDateTime, and OffsetDateTime.
- All of these use the ISO calendar system.

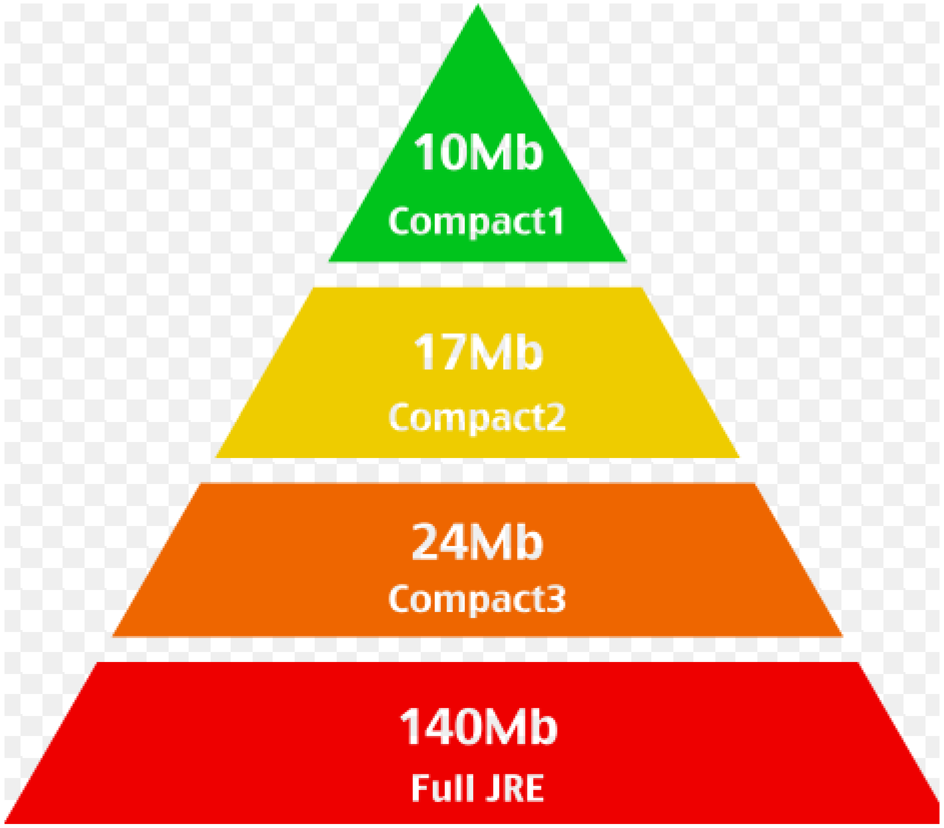
DateTime Sample

- `LocalDate today = LocalDate.now();`
- `LocalDate payday = today.with(TemporalAdjuster.lastDayOfMonth()).minusDays(2);`
- `LocalDate dateOfBirth = LocalDate.of(1984, Month.MARCH, 15);`

Compact Profiles

- enable reduced memory footprint for applications that do not require the entire Java platform.
- Compact profiles are present in additive layers. This means that each Profile contains all of the APIs in the previous smaller Compact Profiles and adds appropriate APIs on top when used.

Compact Profiles



Compact Profiles

Full SE API	Beans	Input Methods	IDL
	Preferences	Accessibility	Print Service
	RMI-IIOP	CORBA	Java 2D
	Sound	Swing	
	AWT	Drag and Drop	
	Image I/O	JAX-WS	
compact3	Security[1]	JMX	JNDI
	XML JAXP[2]	Management	Instrumentation
compact2	JDBC	RMI	XML JAXP
compact1	Core (java.lang.*)	Security	Serialization
	Networking	Ref Objects	Regular Expressions
	Date and Time	Input/Output	Collections
	Logging	Concurrency	Reflection
	JAR	ZIP	Versioning
	Internationalization	JNI	Override Mechanism
	Extension Mechanism	Scripting	

Internationalization

- The JDK 8 release includes support for Unicode 6.2.0.
  - 733 new characters including Turkish Lira sign
  - 7 new scripts:
- The Unicode Consortium has released the Common Locale Data Repository (CLDR) project to "support the world's languages, with the largest and most extensive standard repository of locale data available." The CLDR is becoming the de-facto standard for locale data.