# The Forgotten API

## JSR 10

# Agenda

- Alternatieven
- Preference API
- Backing Stores
- Roadmap
- Referenties

# Alternatieven

- Property files
- OSGi Preferences
- JNDI

# Alternatieven – Property files

Package: java.util.Properties

Voordelen:

- Bekend / wijd verspreid
- Simpel

Nadelen:

- Geen notificatie mechanisme
- Platte oplossing (geen hiërarchie)
- Niet gestandaardiseerd (bestandsnamen, locatie)
- Geen onderscheid tussen systeem en gebruikersinstellingen
- Niet Back End neutraal (bestandsysteem gebonden)

# Alternatieven – OSGi Preferences

Package: org.osgi.service.prefs

Voordelen:

- Simpel
- Vergelijkbaar met de standaard Java Preferences API
- Back End neutraal

Nadelen:

- Geen notificatie mechanisme
- Gebruik van OSGi is een voorwaarde

# Alternatieven – JNDI

Package: javax.naming

Voordelen:

- Back End neutraal
- Notificatie mechanisme

Nadelen:

- Complex ( 5 packages, 80+ classes )
- Geen gestandaardiseerde instellingen policy

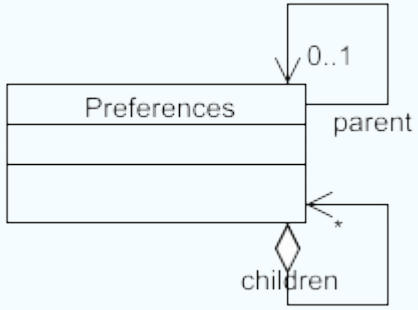
# Preferences API

- Algemeen
- Basis
- Notificatie mechanisme

# Preferences API – Algemeen

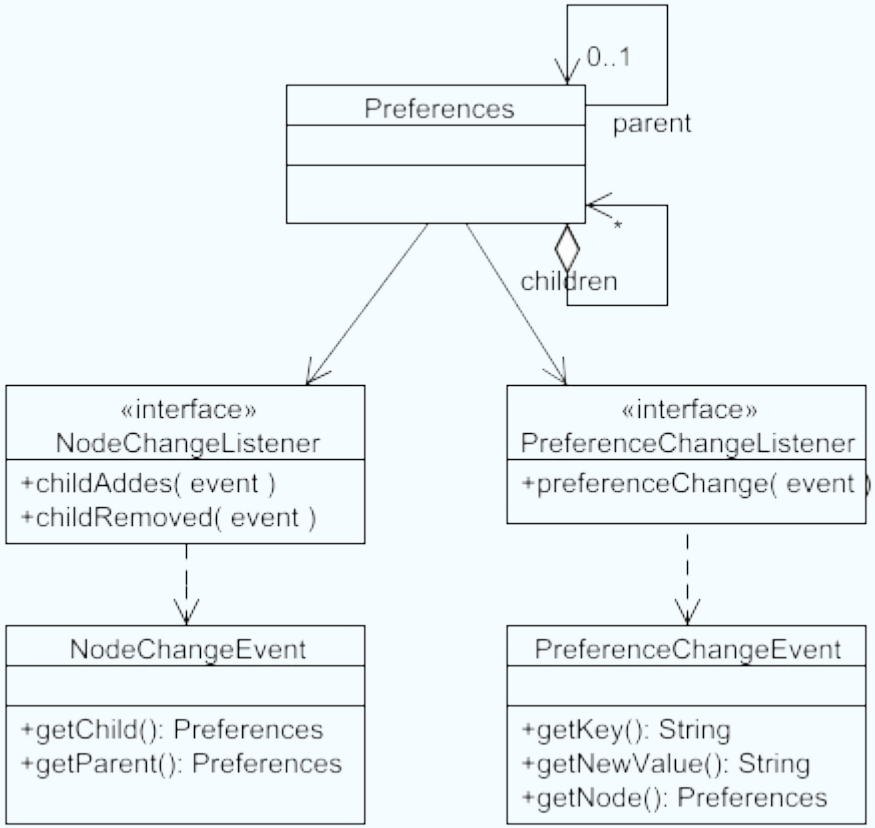
- Beschreven in JSR 10. Final sinds mei 2002
- Standaard onderdeel van Java sinds 1.4
- Package java.util.prefs
- Eenvoudig (3 interfaces en 6 classes)
- Features:
  - Hiërarchische opslag van instellingen, boom-structuur
  - User specifieke instelling naast systeem instellingen
  - Notificatie mechanisme (Observer/Listener pattern)
  - Xml import / export
  - Back End neutral, support voor verschillende backing stores.

# Preferences API – Basis




```
// Node bepalen.
Preferences node;
node = Preferences.systemRoot().node( "/a/b/c" );
node = Preferences.systemNodeForPackage( MyClass.class );
node = node.node( "childNaam" );

// Instellingen lezen
String strInstelling = node.get( "instellingNaam1", "default" );
int intInstelling    = node.getInt( "instellingNaam2", 2 );
...

// Instellingen schrijven
node.put( "instellingNaam1", "waarde" );
node.putInt( "instellingNaam2", 1 );
...
```

# Preferences API – Notificatie



# Preferences API – Notificatie

```
public class InstellingGebruiker implements PreferenceChangeListener {

	private static final String MIJN_INSTELLING = "instellingNaam";
	private String instelling;

	...

	// ---------------------------------------------------------------
	// PreferenceChangeListener implementation
	// ---------------------------------------------------------------

	public void preferenceChange(PreferenceChangeEvent evt) {
		// Is dit mijn instelling?
		if ( evt.getKey().equals( MIJN_INSTELLING ) ) {
			// Verwerk gewijzigde instelling.
			instelling = evt.getNewValue();
			verwerkWijziging();
		}
	}
}
```

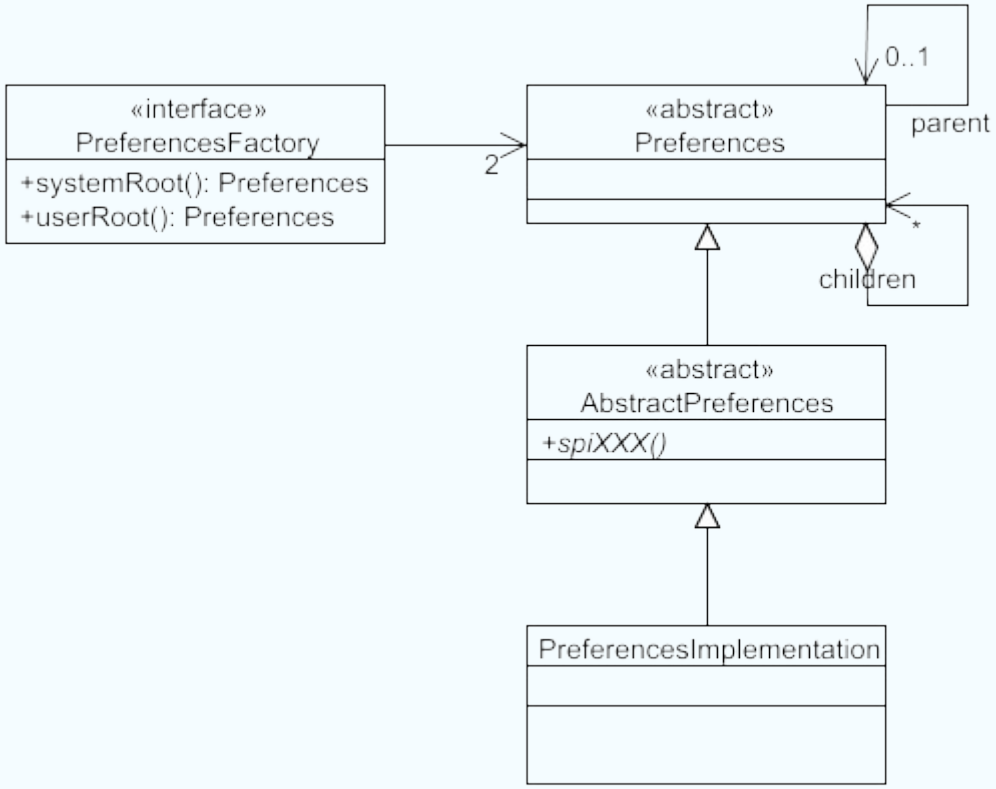
# Preferences API – Notificatie

```
public class InstellingGebruiker implements NodeChangeListener {

	...

	// ----------------------------------------------------------------------
	// NodeChangeListener implementation
	// ----------------------------------------------------------------------

	public void childAdded(NodeChangeEvent evt) {
		Preferences nieuwKind = evt.getChild();
		verwerkNieuwKind( nieuwKind );
	}

	public void childRemoved(NodeChangeEvent evt) {
		String kindNaam = evt.getChild().name();
		verwijderKind( kindNaam );
	}


}
```

# BackingStore

- Algemeen
- Standaard implementatie
- In memory
- Database
- Composite
- Java Content Repository (JCR)
- Hazelcast

# BackingStore




Voordeel:

- Service Provider Interface (SPI), dus meerdere implementaties mogelijk
- Eenvoudig te configureren (`java.util.prefs.PreferencesFactory`)

Nadeel:

- Slechts 1 backing store mogelijk

# BackingStore – Standaard

Voordelen:

- Geen extra actie/configuratie vereist

Nadelen:

- JVM instantie gebonden, sync() mogelijk
- Niet geïsoleerd (1 grote bak per machine voor alles)
- Geen clustering mogelijk (lokale opslag)
- Moeilijk te controleren / beheren

# BackingStore – Memory

Voordelen:

- Simpel, vereist geen speciale eigen configuratie
- Geïsoleerd
- Bruikbaar voor unit testing

Nadelen:

- Instellingen eerst laden voor gebruik
- Geen persistency, wijzigingen gaan verloren bij shutdown
- JVM instantie gebonden

# BackingStore – Database

Voordelen:

- Geïsoleerd
- Gecentraliseerde opslag

Nadelen:

- JVM instantie gebonden, sync() mogelijk

# BackingStore – Composite

Voordelen:

- Geïsoleerd
- Meerdere backing store tegelijk bruikbaar

Nadelen:

- Extra configuratie vereist
- Complexiteit verhogend
- Geen overlay (samenvoegen van node)

# BackingStore – JCR

Voordelen:

- Geïsoleerd
- Clustering (JCR implementatie afhankelijk)
- Niet JVM instantie gebonden, geen sync() nodig
- Gecentraliseerde opslag

Nadelen:

- Extra service vereist

# BackingStore – HazelCast

Voordelen:

- Geïsoleerd
- Clustering
- Niet JVM instantie gebonden, geen sync() nodig

Nadelen:

- Instellingen laden voor gebruik
- Geen persistency, wijzigingen gaan verloren bij shutdown

# Roadmap

- Commandline interface
- C/C++ implementatie
- DotNet implementatie
- OSGi bundles
- Generic Preference Viewers
  - Rich/fat client
  - Web client
  - IDE plugin (Eclipse)

# Referenties

- JSR 10, http://jcp.org/en/jsr/detail?id=10
- OSGi, http://www.osgi.org/javadoc/r4v41/
- Preferences API, http://java.sun.com/j2se/1.4.2/docs/guide/lang/preferences.html
- Hazelcast, http://www.hazelcast.com/