Extendable TypeConfiguration

Our goal is to define a single place where all type information will reside. We define TypeConfiguration class which will have multiple type extensions for different features. This configuration will be set for the whole Ignite instance with ability to override it on per-cache level.

TypeConfiguration

```
public class TypeConfiguration implements Serializable {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /**
     * Type name. Can be one of three things:
     * - Simple type name. Use it when type is not in classpath;
     * - Fully qualified type name. Prefer it when type is in classpath to avoid conflicts.
     *   E.g. "my.package.employee.Address", "my.package.organization.Address";
     * - Package wildcard. Must ends with ".*" in this case. E.g. "my.package.*".
     */
    private String typeName;

    /** Used to configure single type when it is not in classpath. */
    public void setTypeName(String);

    /** Used to configure specific class. Both typeName and packageName fields will be set. */
    public void setClass(Class);

    /** Affinity key field name. */
    private String affKeyFieldName;

    /** Type info extensions. */
    private Map<Class<? extends TypeInfo>, ? extends TypeInfo> typeInfos;

    public TypeInfo[] getTypeInfo() {...}
    public void setTypeInfo(TypeInfo... typeInfos) {...}

    public <T extends TypeInfo> T getTypeInfo(Class<T> infoCls) {...}
}
```

Notes:

- TypeInfo's are set as varargs to follow general Ignite rules (e.g. IgniteConfiguration.setCacheConfiguration()).
- TypeInfo getter/setter do not have "s" at the end to follow general Ignite rules (e.g. IgniteConfiguration.setCacheConfiguration());
- Shouldn't we move "affKeyFieldName" to another TypeInfo, e.g. AffinityKeyTypeInfo?

TypeInfo

```
public interface TypeInfo extends Serializable {
    /** Whether implemenation supports single type. */
    bool supportSingleType();

    /** Whether implementation supports multiple types. */
    bool supportMultipleTypes();
}
```

Notes:

- Package configuration are only supported by PortableTypeInfo for now. For this reason it makes sense to add "support*" methods to prevent misconfiguration.

PersistenceTypeInfo

```
public class PersistenceTypeInfo implements TypeInfo {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** Schema name in database. */
    private String dbSchema;

    /** Table name in database. */
    private String dbTbl;

    /** Persisted fields. */
    private Collection<PersistenceField> fields;
}
```

```
public class PersistenceField implements Serializable {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** Column name in database. */
    private String dbFieldName;

    /** Column JDBC type in database. */
    private int dbFieldType;

    /** Field name in java object. */
    private String javaFieldName;

    /** Corresponding java type. */
    private Class<?> javaFieldType;
}
```

Notes:

- Package wildcards are not supported.

PortableTypeInfo

```
public class PortableTypeInfo implements TypeInfo {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** ID mapper. */
    private PortableIdMapper idMapper;

    /** Serializer. */
    private PortableSerializer serializer;

    /** Portable metadata enabled flag. When disabled queries and pretty toString() will not work. */
    private Boolean metaDataEnabled = true;
}
```

Notes:

- Supported for both types and package wildcards.

QueryTypeInfo

```
public class QueryTypeInfo implements TypeInfo {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** Fields. */
    private QueryField[] flds;

    /** Group indexes. */
    private QueryCompoundIndex[] grpIdxs;

    /** Fields to index as text. */
    private String[] txtIdxs;
}
```

```
public class QueryField implements Serializable {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** Field name. */
    private String name;

    /** Field class. */
    private Class cls;

    /** Whether to index the field. Disabled by default. */
    private QueryFieldIndexType idxTyp = QueryFieldIndexType.NONE;
}
```

```
public enum QueryFieldIndexType {
    /** Do not index the field. */
    NONE,

    /** Ascending order. */
    ASC,

    /** Descending order. */
    DESC
}
```

```
public class QueryCompoundIndex implements Serializable {
    /** Serial version UID. */
    private static final long serialVersionUID = 0L;

    /** Index name. */
    String name;

    /** Participating fields. */
    private QueryFieldInfo[] flds;
}
```

Notes

- QueryFieldInfo.cls is a problem for non-Java users, because they have to write ugly things like "java.lang.Integer" which is very hard to understand for non-Java users. Lets switch to enum here?

IgniteConfiguration

```
public class IgniteConfiguration {
    /** Type configurations. */
    private TypeConfiguration[] typeCfg;

    public TypeConfiguration[] getTypeConfiguration();
    public void setTypeConfiguration(TypeConfiguration... typeCfg);
}
```

CacheConfiguration

```
public class CacheConfiguration {
    /** Type configurations. */
    private TypeConfiguration[] typeCfg;

    public TypeConfiguration[] getTypeConfiguration();
    public void setTypeConfiguration(TypeConfiguration... typeCfg);
}
```