JAVA 2

Reflection

Teme

- *Class* objekt
- *ClassLoader*
- Reflection API
- *Annotations*
- *Factory Pattern*

Class objekt

- JRE posjeduje sve informacije o klasama kroz *immutable Class* objekt
- sadrži metode za dohvat varijabli instance, metoda, konstruktora, konstanti...
- pristup
  - *"milica".getClass()* – svaki objekt *vidi* svoju klasnu reprezentaciju
  - *java.lang.String.class* – svaka klasa može pristupiti klasnoj reprezentaciji
  - *Class.forName("java.lang.String")* – dinamičko učitavanje klase u JVM
- omogućuje dinamičko kreiranje objekata
  - *java.lang.String.class.newInstance()*

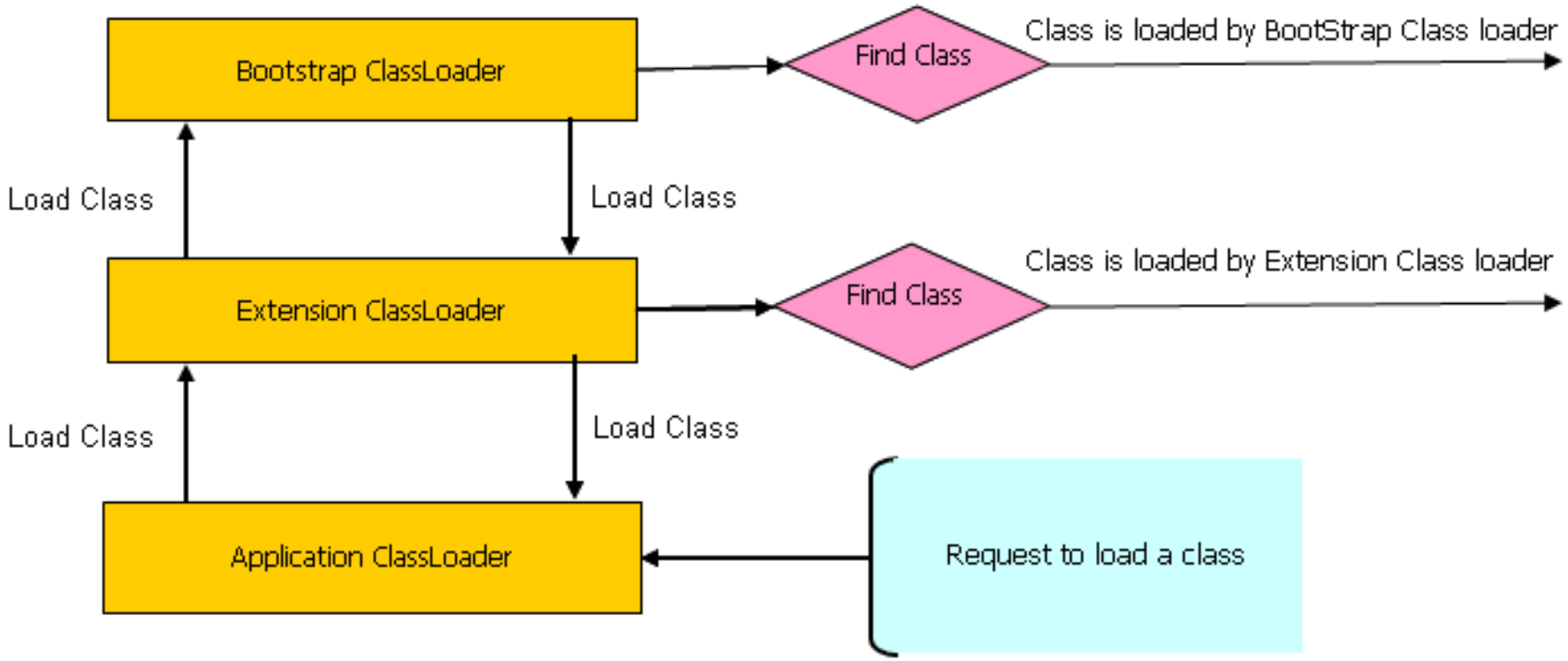
ClassLoader

- dinamički učitava klase u JVM
- *Bootstrap ClassLoader* – napisan u nativnom jeziku
  - bootstrapping problem - njegovu klasu JVM učitava sam
  - učitava standardne pakete JRE (runtime libs - java.lang, java.util...), prilikom startanja JVM
- *Extension(Platform) ClassLoader*
  - učitava klase definirane *java.ext.dirs* svojstvom ($JAVA_HOME/jre/lib/ext)
  - *Bootstrap* mu je *parent*
- *Application(System) ClassLoader*
  - učitava klase koje su definirane putanjom klasa
    - *classpath environment variable*
    - *-classpath ili -cp command line option*
  - *Platform* mu je *parent*

ClassLoader

- principi rada
  - *delegation* – kada aplikacija zatraži klasu, definiranu *classpath-om*, *System* će delegirati *Platformu,* a ovaj *Bootstrapu* – *ako* nitko od *parenta* nije učitao, *System* učitava klasu
  - *uniqueness* – sve klasu su jedinstvene - logična posljedica *delegationa*
  - *visibility* – klase učitane od *parenta* su vidljive *childovima*, ali ne i *vice versa*
- implementacija vlastitog *ClassLoadera*
- važno je razlikovati učitavanje klase (*load*) od njene inicijalizacije (*initialization*) kada se izvršavaju svi *static* blokovi

ClassLoader



Izvor: https://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html

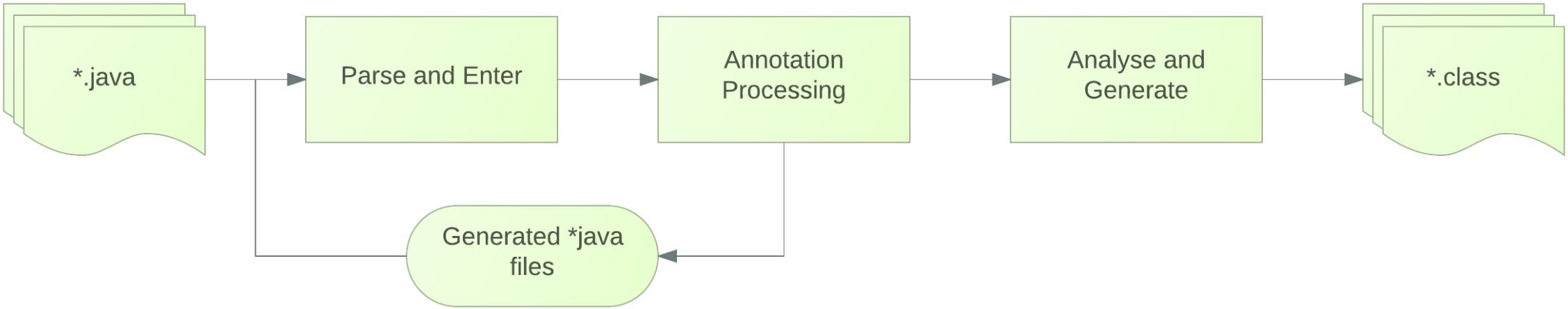
Reflection API

- izuzetno moćan, napredan način korištenja Java jezika
- istraživanje klasa, sučelja, varijabli, anotacija, metoda (...) tijekom izvršavanja aplikacije
- dinamičko kreiranje objekata, prema imenu
- dinamičko pozivanje metoda
- izgradnja *debugger*-a, GUI editora, mapiranje JSON sa atributima klasa...

Annotations

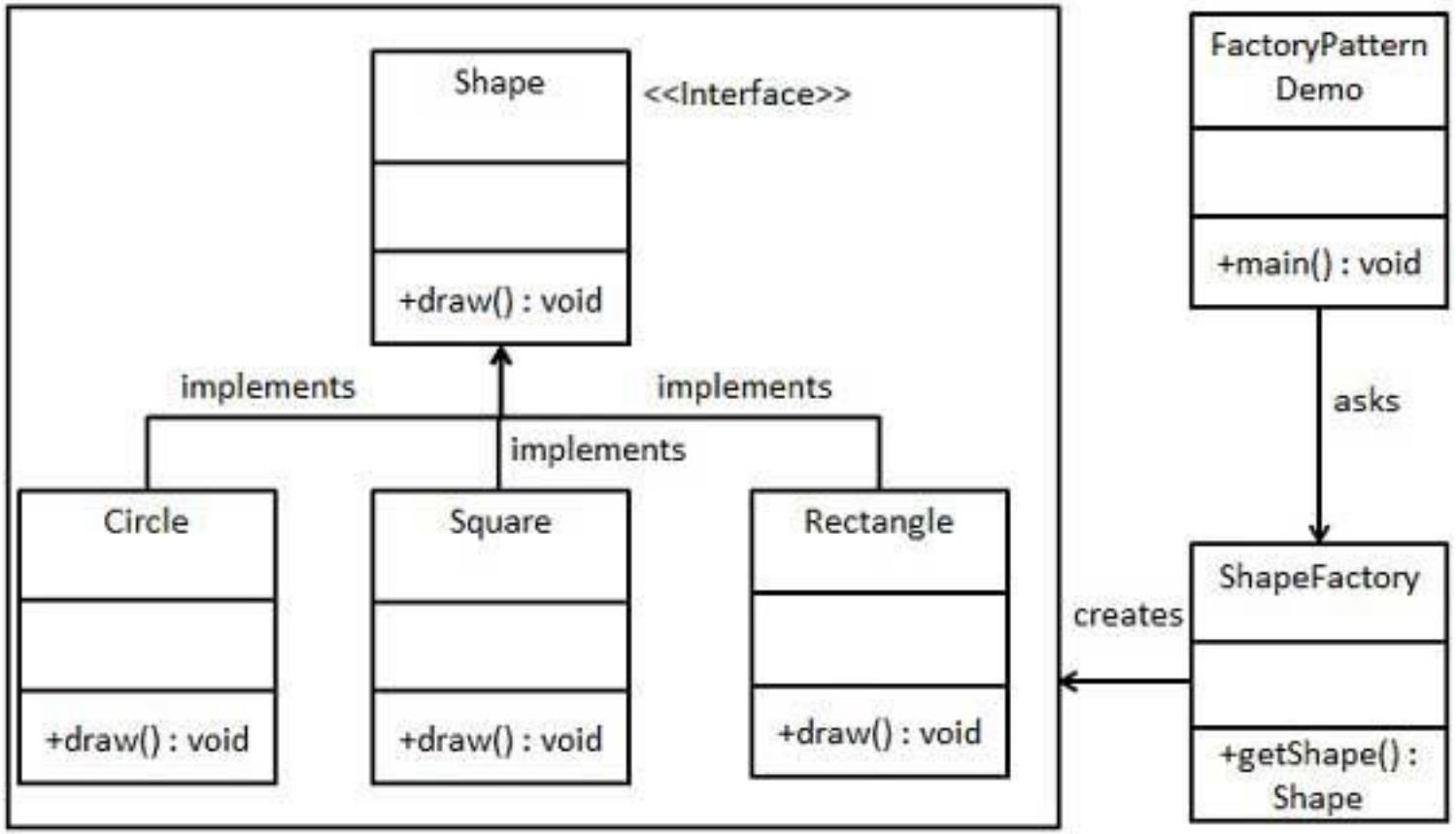
- *metadata* – informacije o kodu, nemaju efekt nad kodom
- anotacije mogu biti anotirane
- @Retention
  - SOURCE – analizirani kompajlerom i ne pohranjuju se u *bytecode*
  - CLASS – pohranjuju se u *bytecode,* ali ne mogu se dohvatiti tijekom izvršavanja
  - RUNTIME - pohranjuju se u *bytecode,* refleksijom mogu se dohvatiti tijekom izvršavanja
- @Target – definiraju što mogu anotirati
  - TYPE, METHOD, FIELD...
- @Interited – automatsko nasljeđivanje
- vrlo popularne u Android razvoju – omogućuju izbjegavanje *boilerplate* koda i redundancija

Annotations



Izvor: https://medium.com/@lgvalle/how-butterknife-actually-works-85be0afbc5ab

Factory Pattern



Izvor: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Hvala na pažnji!