# Tölvunarfræði 1
## Fyrirlestur 15: Forritasöfn í Java I

Hjálmtýr Hafsteinsson

Haust 2015

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

# Í síðasta fyrirlestri

- Skipulag forrita með föllum
- Stærri forritadæmi
- Notkun fylkja í föllum
  - Fylki sem viðföng
  - Fylki sem skilagildi

**Kafli 2.1**

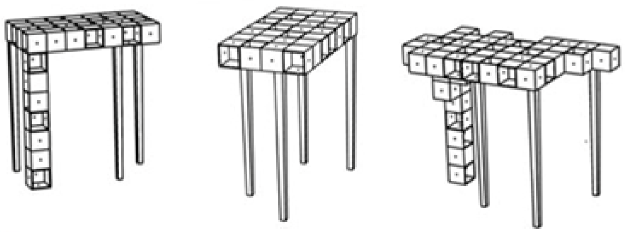
# Í þessum fyrirlestri

- Forritasöfn (*libraries*)
  - Skipulagning falla í forritasöfn
  - Forritaskil (*API*)
- Dæmi um forritasöfn
  - **`Gaussian`**
  - **`StdRandom`**

**Kafli 2.2**

# Skipulag Java forrita

- Hingað til hafa forritin okkar öll verið í einni **`.java`** skrá
  - Þetta er óþarflega takmarkandi
  - Auðvelt að nota föll úr öðrum skrám
  - Java byggir mikið á að nota forritasöfn (*libraries*)
- Að nota föll á milli **`.java`** skráa:
  - Leyfir okkur að útvíkka Java forritunarmálið
  - Gerir einingarforritun (*modular programming*) mögulega



Forritið sett saman úr sjálfstæðum einingum sem vinna saman

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI- OG TÖLVUNARFRÆÐIDEILD

# Að nota föll úr öðrum `.java` skrám

- Höfum þegar notað föll úr öðrum skrám:
  - **`Math`**, **`StdIn`**, **`StdDraw`**, …
- Til að vísa til falla í öðrum skrám þarf:
  - Báðar skrár í sömu möppu
  - Til að kalla á fall, setja klasanafn á undan og punkt á milli
    - Til dæmis **`StdOut.printf()`**

Eða aðgengilegar með CLASSPATH umhverfisbreytunni

# Dæmi um samskipti á milli `.java` skráa

- Í tilteknu samræmdu prófi (*SAT*) er meðeinkunn 1019 og staðalfrávikið 209
  - Viljum fá að vita hve mörg prósent voru fyrir neðan okkar einkunn
  - Skrifum forritið **`SATmyYear.java`** til að finna það út:

```
public class SATmyYear {
   public static void main(String args[]) {

      double z = Double.parseDouble(args[0]);
      double v = Gaussian.Phi((z-1019)/209);

      StdOut.println(v);
   }
}
```

Köllum á fallið **`Phi`** í **`Gaussian`** klasanum

Þurfum að hafa skránna **`Gaussian.java`** í sömu möppu

# Hvað gerist í `Gaussian.java`?

```
public class Gaussian {
   ...
   public static double phi(double x) {
      return Math.exp(-x*x / 2) / Newton.sqrt(2 * Math.PI);
   }

   public static double Phi(double z) {
      if (z < -8.0) return 0.0;
      if (z >  8.0) return 1.0;
      double sum = 0.0, term = z;
      for (int i = 3; sum + term != sum; i += 2) {
         sum  = sum + term;
         term = term * z * z / i;
      }
      return 0.5 + sum * phi(z);
   }
   ...
}
```

Í upphaflega **`Gaussian.java`** var kallað á **`Math.sqrt`**, en við skulum nota fallið **`sqrt`** úr **`Newton.java`**

**`SATmyYear`** kallar á fallið **`Phi`**

**`Phi`** kallar á fallið **`phi`**

Fallið **`phi`** kallar svo á fallið **`sqrt`** í **`Newton.java`**

# Heildarmyndin

SATmyYear.java

```
public class SATmyYear
{
   public static void main(String[] args)
   {
      double z = Double.parseDouble(args[0]);
      double v = Gaussian.Phi((z - 1019)/209);

      StdOut.println(v);
   }
}
```

Newton.java

```
public class Newton
{
   public static double sqrt(double c)
   {
      if (c < 0) return Double.NaN;
      double err = 1e-15;
      double t = c;
      while (Math.abs(t - c/t) > err * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i]);
         StdOut.println(x);
      }
   }
}
```

Gaussian.java

```
public class Gaussian
{
   public static double Phi(double z)
   {
      if (z < -8.0) return 0.0;
      if (z >  8.0) return 1.0;
      double sum = 0.0, term = z;
      for (int i = 3; sum != sum + term; i += 2)
      {
         sum  = sum + term;
         term = term * z * z / i;
      }
      return 0.5   +  phi(z)   * sum;
   }

   public static double phi(double x)
   {
      return Math.exp(-x*x/2) /
         Newton.sqrt(2*Math.PI);

   }

   public static void main(String[] args)
   {
      double z     = Double.parseDouble(args[0]);
      double mu    = Double.parseDouble(args[1]);
      double sigma = Double.parseDouble(args[2]);
      StdOut.println(Phi((z - mu) / sigma));
   }
}
```

# Nokkur atriði

- Lykilorðið **`public`**
  - Tilgreinir að fall í klasa er aðgengilegt úr öðrum forritum
  - Ef við viljum það ekki þá notum við lykilorðið **`private`**

```
private static int falid(int i) {
    ...
}
```

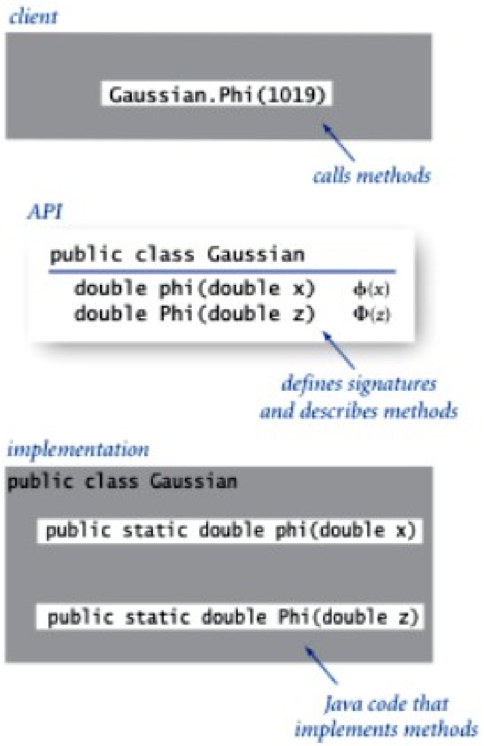
- **`.class`** skrár
  - Í Java inniheldur hver **`.java`** skrá einn klasa (*class*)
  - Þegar hún hefur verið þýdd (með **`javac`**) þá verður til **`.class`** skrá sem inniheldur einn klasa
  - Þurfum aðeins að hafa aðgang að **`.class`** skránum til að geta notað föllin úr klasanum

# Fleiri atriði

- Klasar eru þýddir þegar það þarf
  - Þegar við þýðum forrit þá þýðir javac allar þær skrá sem þarf að þýða til að hægt sé að keyra forritið
  - Þær skrá sem ekkert hafa breyst eru ekki þýddar
    - Þýðandinn ber saman tímasetningu á **`.java`**-skrá og **`.class`**-skrá
  - Forritasöfn eins og **`Math`**, **`StdIn`**, **`Gaussian`** breytast aldrei (eða mjög sjaldan!)
- Mörg **`main`**-föll
  - Hver klasi hefur eitt **`main`**-falli
  - Þegar við keyrum forrit (með: **`java`** ***`xxx`***) þá hefst keyrsla í **`main`**-falli <u>þess klasa</u>

# Forritasöfn (*libraries*)

- Klasi sem inniheldur föll sem önnur forrit nota kallast forritasafn
- Hugtök tengd forritasöfnum:
  - Notandi, biðlari (*client*):
    - Forritið sem kallar á fallið
  - Forritaskil (*API*):
    - "Samningur" milli notanda og útfærslu um hvað fallið eigi að gera (*Application Programming Interface*)
  - *Útfærsla* (*implementation*)
    - Java kóði sem útfærir fallið í forritasafninu



Library abstraction

# Dæmi um forritasafn: `Gaussian`

Forritaskil fyrir **`Gaussian`**

| public class Gaussian | |
|---|---|
| double phi(double x) | $\phi(x)$ |
| double phi(double x, double m, double s) | $\phi(x, \mu, \sigma)$ |
| double Phi(double z) | $\Phi(z)$ |
| double Phi(double z, double m, double s) | $\Phi(z, \mu, \sigma)$ |

Þetta er lýsingin á hvað fallið gerir

Útfærsla á **`Gaussian`**

```
public class Gaussian {

   public static double phi(double x) {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

   public static double phi(double x, double mu, double sigma) {
      return phi((x - mu) / sigma) / sigma;
   }

   ...

}
```

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLA
OG TÖLVUNARFRÆÐIDEILD

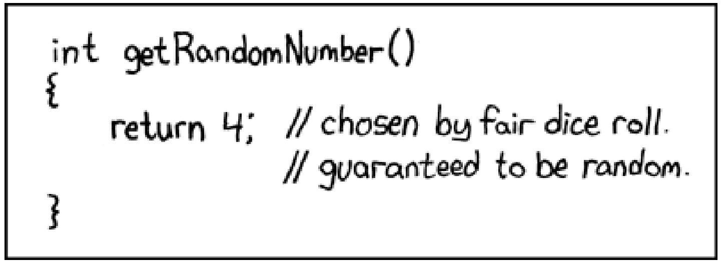
# Fyrirlestraræfing

1. Nefnið helstu kosti einingaforritunar?
2. Sum forritasöfn leyfa forriturum ekki að sjá útfærslu fallanna (þ.e. `.java` skránna), heldur bara forritaskilin (*API*). Hver gæti verið ástæðan fyrir því?
3. Væri hægt að kalla á `main`-fall í öðrum klasa? Hvernig væri það þá gert?

# Slembitölur (*random numbers*)

- Oft gott að prófa forrit með tilbúnum gögnum
- Ein stök tala er ekki slembin
  - Er 4 slembin? En 9?
- Getum leitt líkur að því að röð talna sé slembin
  - Tölurnar eru "rétt" dreifðar og ekki of mikil fylgni
  - Til sérstök tölfræðipróf fyrir slembitölur
    - Sumar skattstofur nota tölfræðipróf til að kanna hvort tölur í skattframtölum séu skáldaðar

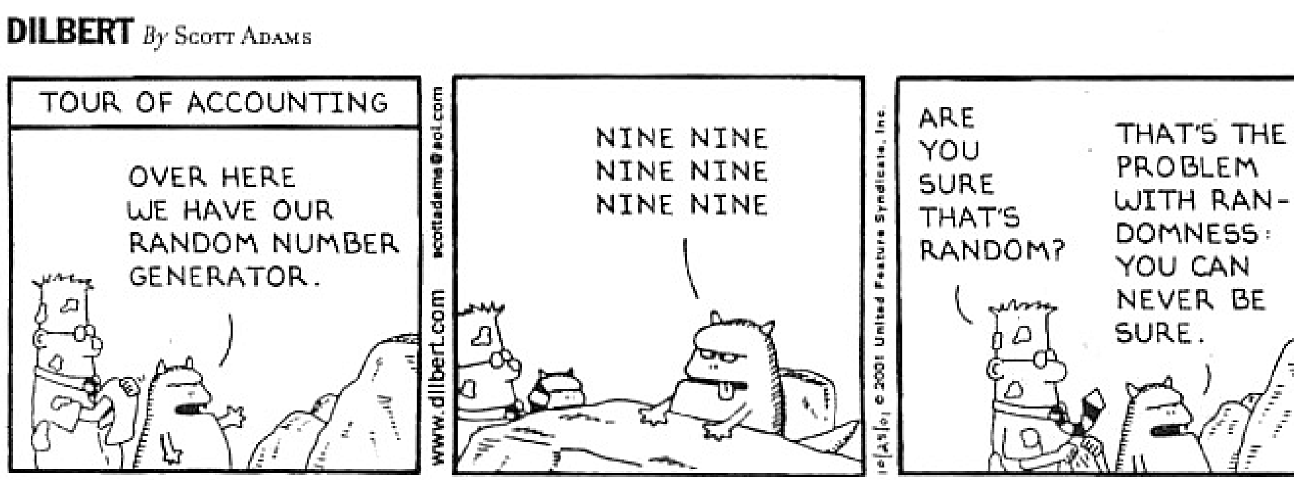
xkcd.com



# Slembitölugjafar

Sjá [random.org](random.org)

- Erfitt að finna "raunverulegar" slembitölur
  - Mæla ýmis eðlisfræðileg fyrirbæri sem eru ófyrirsjáanleg
  - Til dæmis: suð í andrúmslofti, úttak úr Geiger-teljara, ...
- Við notum reiknirit (*algorithms*) til að búa til slembitölur
  - Oft kallaðar gervislembitölur (*pseudo-random*)



# Slembitölugjafar í tölvum

- Nota stærðfræðiformúlu sem býr til næstu slembitölu út frá þeirri síðustu
  - Dæmi:

$$r_{n+1} = (3 * r_n + 4) \textbf{ mod } 10$$

Þessi aðferð kallast línuleg samleifaraðferð (*linear congruential method*)

setjum $r_0 = 0$

þá er $r_1 = (3*0 + 4) \textbf{ mod } 10 = 4$

$r_2 = (3*4 + 4) \textbf{ mod } 10 = 6$

$r_3 = (3*6 + 4) \textbf{ mod } 10 = 2$

$r_4 = (3*2 + 4) \textbf{ mod } 10 = 0$

Erum nú komin í hring. Fáum alltaf sama hringinn

Í raunverulegum slembigjöfum er hringurinn mun lengi og við fáum öll möguleg gildi

# Slembitölur í Java

- Höfum séð fallið **`Math.random()`**
  - Skilar slembitölum á bilinu [0.0 1.0)
  - Byggir á Java klasanum **`Random`**
- Java slembitöugjafinn notar línulega samleifaraðferð
  - Vinnur með **`long`** gildi (þ.e. 64-bita heiltölur)

$$\boxed{r_{n+1} = (25.214.903.917 * r_n + 11)\ \mathbf{mod}\ 2^{48}}$$

Virðist þokkalega góður slembitölugjafi samkvæmt prófunum

  - Býr bara til 48-bita heiltölu í hvert sinn
  - Henni breytt í **`double`** tölu fyrir **`Math.random()`**
  - Notar fjölda nsek frá 1. jan. 1970 sem upphafsgildi (*seed*)

Pláss fyrir um 292 ár
(≈ $2^{63}$ nanósek)

# Forritasafnið `StdRandom`

- Smíðum okkar eigið forritasafn til að búa til slembitölur af ýmsum gerðum

`public class StdRandom`

| | | |
|---|---|---|
| `int` | `uniform(int N)` | *integer between* `0` *and* `N-1` |
| `double` | `uniform(double lo, double hi)` | *real between* `lo` *and* `hi` |
| `boolean` | `bernoulli(double p)` | `true` *with probability* `p` |
| `double` | `gaussian()` | *normal, mean* `0`*, standard deviation* `1` |
| `double` | `gaussian(double m, double s)` | *normal, mean* `m`*, standard deviation* `s` |
| `int` | `discrete(double[] a)` | `i` *with probability* `a[i]` |
| `void` | `shuffle(double[] a)` | *randomly shuffle the array* `a[]` |

# Nokkur af föllunum í **StdRandom**

```
public class StdRandom {

   // between 0 and N-1
   public static int uniform(int N) {
      return (int) (Math.random() * N);
   }

   // between a and b
   public static double uniform(double a, double b) {
      return a + Math.random() * (b-a);
   }

   // true with probability p
   public static boolean bernoulli(double p) {
      return Math.random() < p;
   }

   // gaussian with mean = 0, stddev = 1
   public static double gaussian()
      /* see Exercise 1.2.27 */

   // gaussian with given mean and stddev
   public static double gaussian(double mean, double stddev) {
      return mean + (stddev * gaussian());
   }

   ...
}
```

Heiltala frá 0 til $N$-1

Heiltala frá $a$ til $b$

Skilar **satt** með líkunum $p$ ($0 \leq p \leq 1$)

Skilar Gauss-dreifðum slemigildum, með $\mu$=0 og $\sigma$=1

Skilar Gauss-dreifðum slemigildum, með gefnum gildum á $\mu$ og $\sigma$

# Einingaprófun (*unit test*)

- Látum fylgja `main`-fall sem prófar öll föllin:

```
public class StdRandom {
   ...

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++) {
         StdOut.printf(" %2d " , uniform(100));
         StdOut.printf("%8.5f ", uniform(10.0, 99.0));
         StdOut.printf("%5b "  , bernoulli(.5));
         StdOut.printf("%7.5f ", gaussian(9.0, .2));
         StdOut.println();
      }
   }
}
```

Viðfang á skipanalínu (`N`) segir hversu oft á að keyra öll föllin

```
% java StdRandom 5
61 21.76541  true 9.30910
57 43.64327 false 9.42369
31 30.86201  true 9.06366
92 39.59314  true 9.00896
36 28.27256 false 8.66800
```

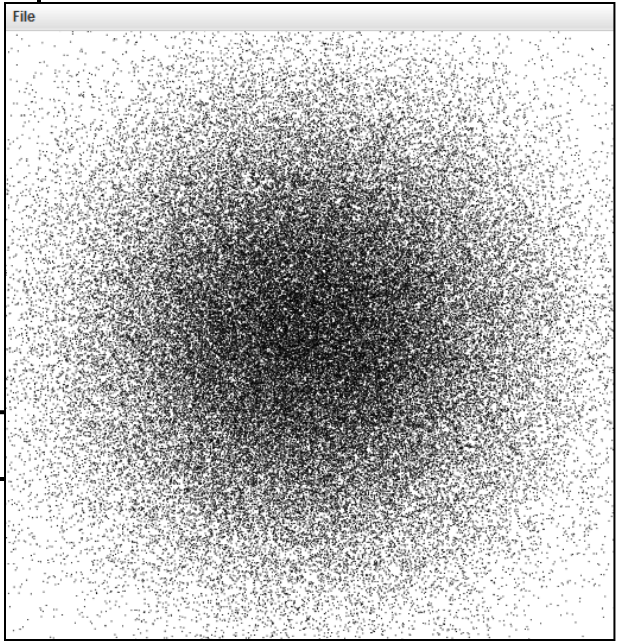
# Dæmi um notkun

- Teikna punkta sem eru dreifðir samkv. Gauss dreifingu

```
public class RandomPoints {
   public static void main(String args[]) {

      int N = Integer.parseInt(args[0]);

      for (int i = 0; i < N; i++) {
         double x = StdRandom.gaussian(0.5, 0.2);
         double y = StdRandom.gaussian(0.5, 0.2);
         StdDraw.point(x, y);

         if (i % 100 == 0) StdDraw.show(10);
      }
   }
}
```

```
% java RandomPoints 100000
```

Einskonar þrívíddar bjölluferill



# Fyrirlestraræfing

4. Hvað gerist í slembitölugjafanum hér á undan ( $r_{n+1}$ = (3*$r_n$ + 4) **mod** 10 ) ef við byrjum með $r_0$ = 3?

5. Eru allar heiltölur frá 0 til 10 jafnlíklegar sem úttak út skipuninni hér fyrir neðan?

   ```
   Math.round(Math.random()*10.0)
   ```

4. Skrifið fallið `int rolldice()`, sem skilar slembiheiltölur frá 1 til 6

# Samantekt

- Í þessum tíma:
  – Forritasöfn
  – Einingaforritun

Kafli 2.2

- Í næsta tíma:
  – Stærri dæmi um forritasöfn
  – Ítruð föll

Kafli 2.2