Bulls and Cows

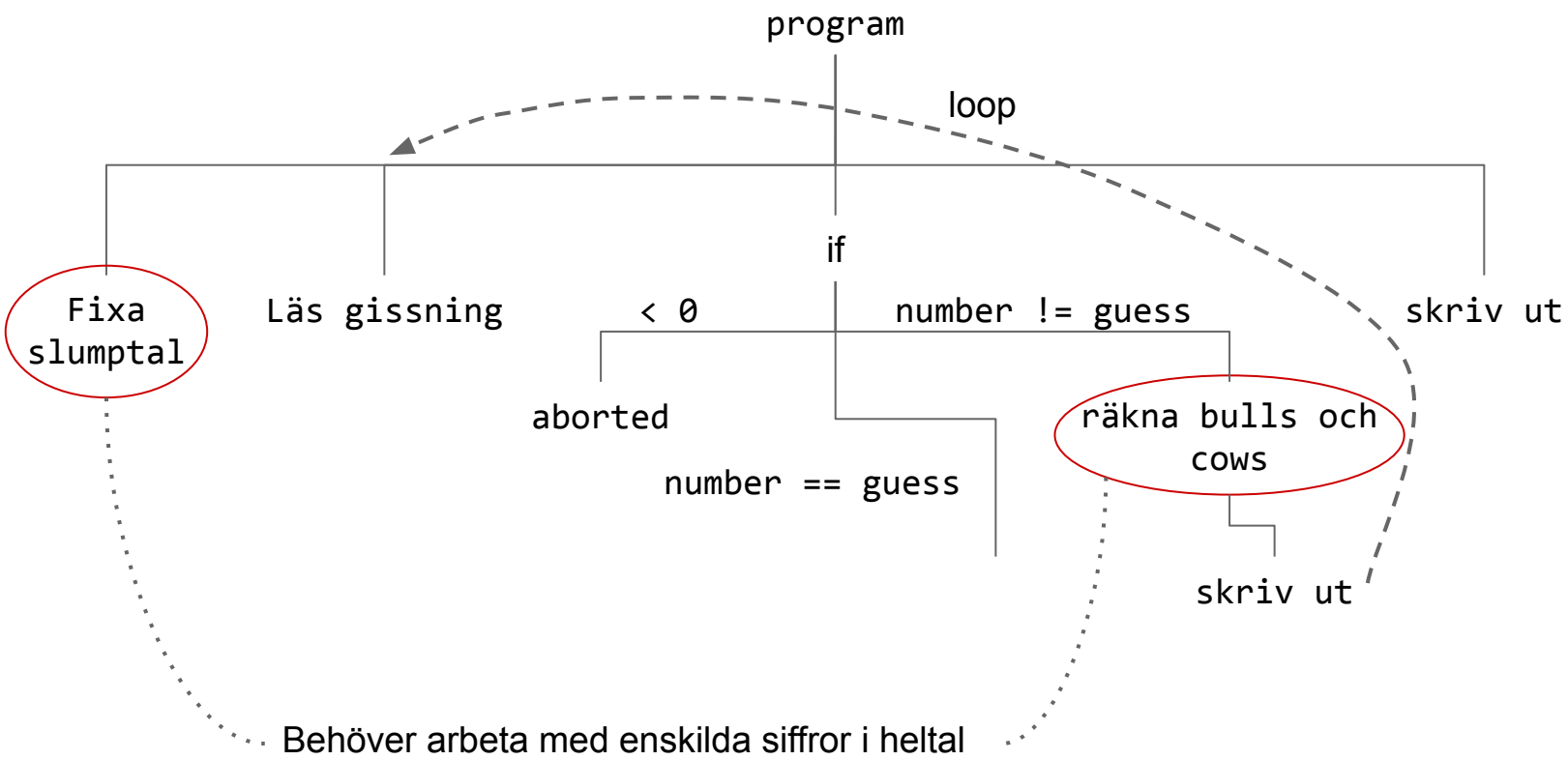
Ett (lite större) programexempel

Joachim von Hacht

Bulls and Cows

```
Welcome to Bulls and Cows.
Try to guess a 4 digit number with digits 1-9
and no repeating digits (-1 to abort).
Bulls = correct digits in correct positions.
Cows = correct digits.
> 5146
There are 0 bull(s) and 2 cow(s)
> 3678
There are 1 bull(s) and 1 cow(s)
> 3845
There are 3 bull(s) and 0 cow(s)
> 3849
There are 2 bull(s) and 0 cow(s)
> 3852
There are 2 bull(s) and 1 cow(s)
> 3815
Done, number was 3815 you needed 6 guesses
```

Skiss


program
loop
Fixa slumptal
Läs gissning
if
< 0
number != guess
skriv ut
aborted
number == guess
räkna bulls och cows
skriv ut
Behöver arbeta med enskilda siffror i heltal

Frågor ...

- Mycket kontroller; unika siffror, inte 0, har cow, har bull
- Hur arbetar man med enskilda siffror?
- Oj, programmet verkar komplicerat, … hur behärska detta ..?

Maska ut Siffror ur Heltal

För att komma åt enskild siffror (från höger) används % (modulo) och / (heltalsdivision)

123 % 10 -> **3**

123 / 10 -> 12

12 % 10 -> **2**

12 / 10 -> **1**

1 % 10 -> 1

1 / 10 -> 0

Metoder

En metoder är avgränsad del av ett program som utför en viss uppgift (för en beräkning t.ex.)

- Metoder använder enkla satser eller if, while, for o.s.v. för att utföra sitt arbete på samma sätt som i själva programmet

Metoder ger oss ingen mer “problemlösningskraft”, de är bara ett sätt att organisera programmet

- När programmen blir större är det väldigt viktig att de kan organiseras på ett begripligt sätt.

Deklarera metoder

Metoder deklarera (skrivs) på samma “nivå” som program(). Vi skriver dem efter program(). Ordningen spelar ingen roll. t.ex

```
int add(int a, int b) {
   return a + b;
}
```

Generellt ordning på deklarationen (v->h, upp->ned)

- Returtyp
- Namn
- Parameterlista i parentes
- Ett block med satser
- Blocket innehåller en return-sats (vanligen)

Anropa metoder

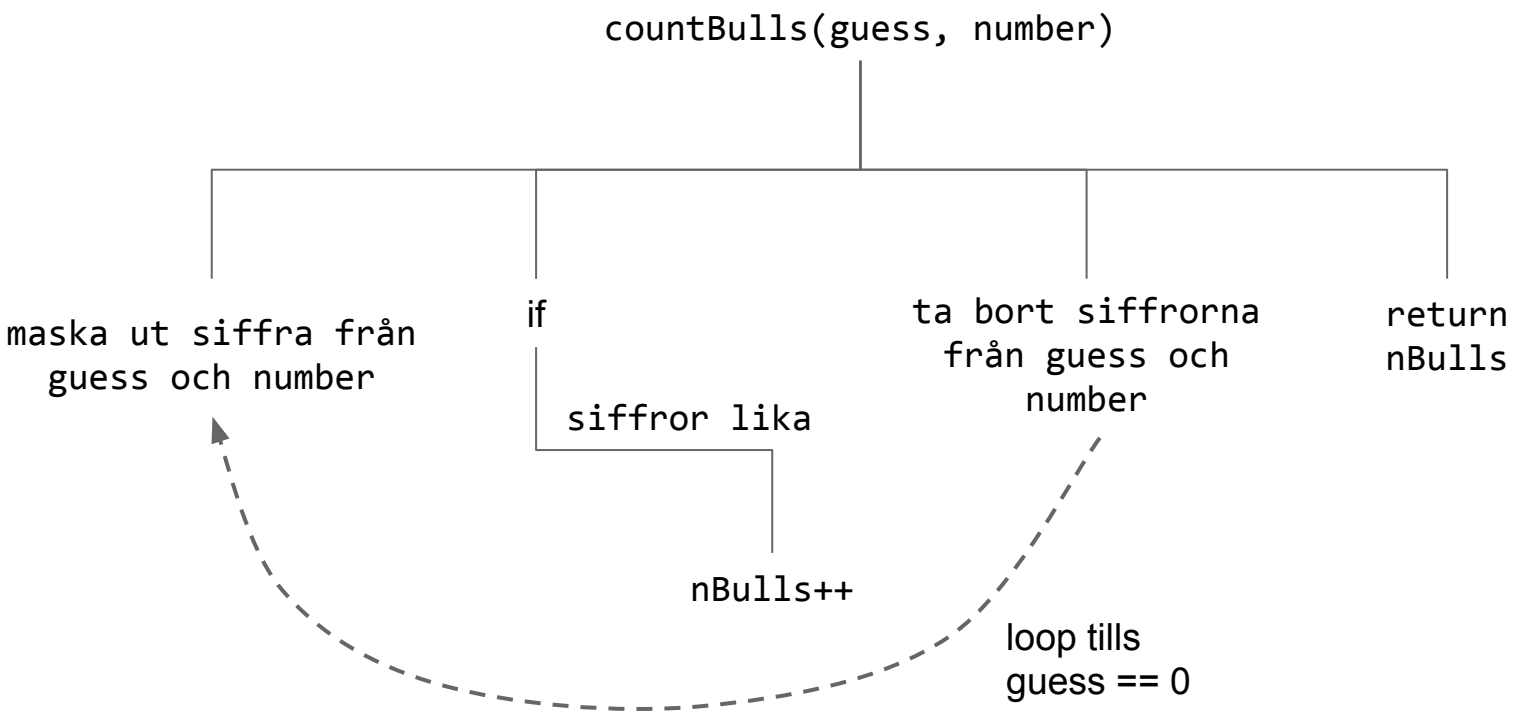
Görs genom att skriva namnet och en parentes med korrekt antal (och typ på) argument

- Vill man spara resultatet måste man göra en tilldelning

```
int add(int a, int b) {  // Declaration
   return a + b;
}

int i = add(1, 3);  // Call add and save result in i
```

Metod CountBulls



Funktionell Abstraktion

Innebär att man antar att man har metod som kan åstadkomma det man vill.

Vi måste bestämma

- Vad metoden måste veta för att kunna utföra sitt arbete (vilka parametrar)
- Vad vi vill ha för svar av metoden (vilken typ av returvärde)

Man fokusera på vad skall göras inte hur!

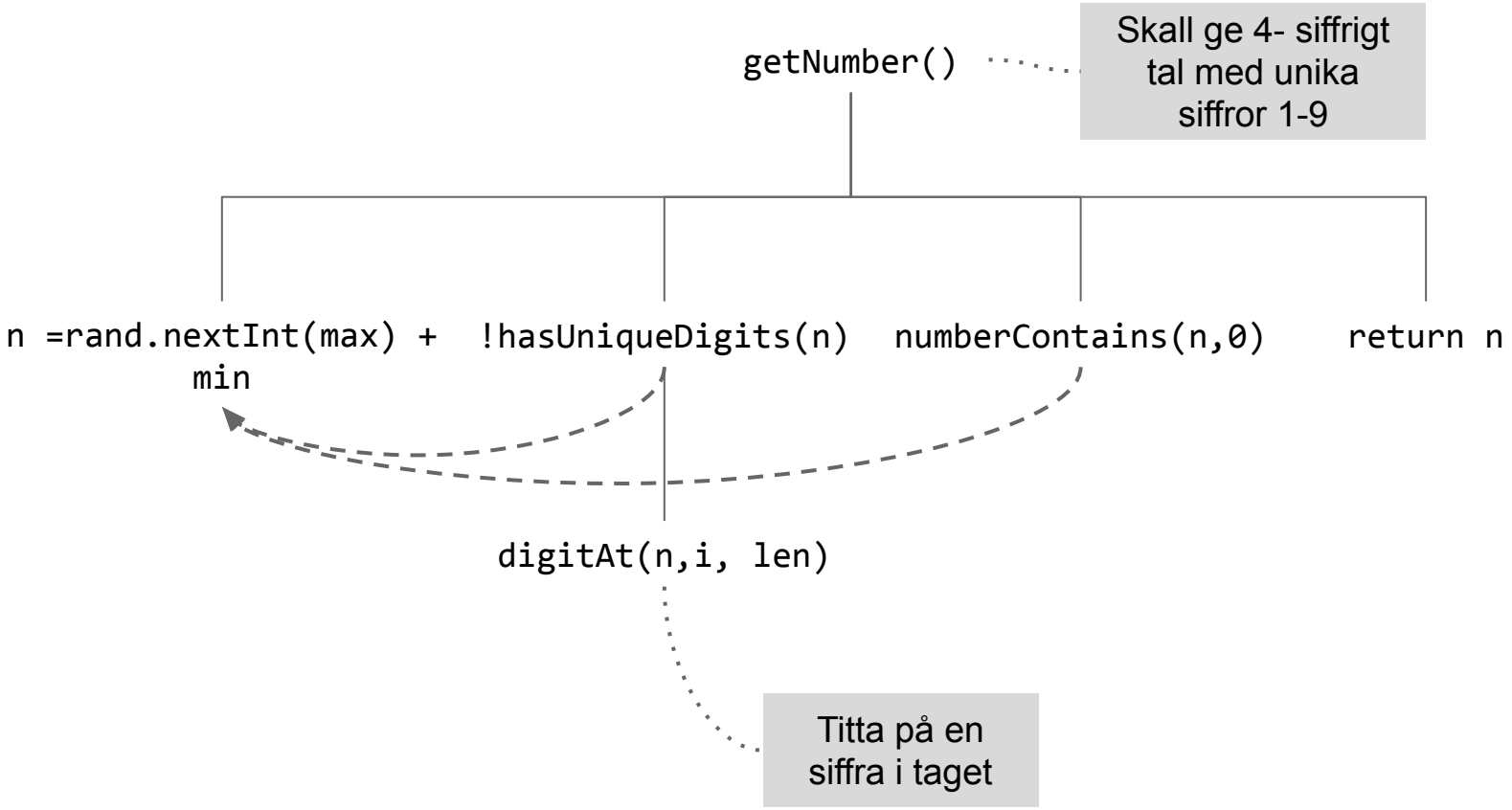
- Vi måste veta vad vi vill göra innan vi gör det.

Funktionell Nedbrytning

För att behärska komplexa problem gör man en funktionell nedbrytning. I princip samma som de diagram vi ritat.

- När vi stöter på en komplex del i programmet antar man att man har en metod som löser problemet (= funktionell abstraktion)
- Kan man inte efter lite eftertanke implementera metoden bryter man ner den och antar att man har ytterligare metoder som kan lösa olika delar …
- … o.s.v. tills man har så enkla metoder som lätt kan implementeras
- Implementera därefter nedifrån (de enkla) och använd dessa för att lösa de svårare problemen.

Funktionell nedbrytning: getNumber



Testning

För att funktionell nedbrytning skall fungera måste vi veta att delarna (metoderna) fungerar.

Detta görs genom att testa metoderna

- I en separat del av programmet …
- … anropar man metoderna med olika varianter av indata och jämför med kända resultat (utdata).
- Jämförelsen skrivs ut (skall skriva true) t.ex testa metod digitAt()

  ```
  out.println(digitAt(12345, 0) == 1);
  ```
  (index 0 är 1, d.v.s. true)
- Genom att skriva ut true behöver vi inte tänka till om resultatet är rätt.

Alla tester sparas,

- Så fort vi ändrat något i programmet kan vi köra alla tester igen!

Programmet

```
void program() {
   out.println("Welcome to Bulls and Cows.");
   out.println("Try to guess a 4 digit number with digits 1-9");
   out.println("and no repeating digits (-1 to abort).");
   out.println("Bulls = correct digits in correct positions.");
   out.println("Cows = correct digits.");

   boolean aborted = false;
   int nGuess = 0;
   int guess;
   int nBulls;
   int nCows;

   int number = getNumber();
   out.println(number);

   while (true) {
       out.print("> ");
       guess = sc.nextInt();
       if (guess < 0) {
           aborted = true;
           break;
       }
       nGuess++;
       if (guess == number) {
           break;
       } else {
           nBulls = countBulls(guess, number);
           nCows = countCowsAndBulls(guess, number) - nBulls;
       }
       out.println("There are " + nBulls + " bull(s) and " + nCows + " cow(s)");
   }
   if (aborted) {
       out.println("Game aborted");
   } else {
       out.println("Done, number was " + number + " you needed " + nGuess + " guesses");
   }
}
```

Lite väl kompakt skrivet p.g.a. utrymme (forts. nästa bild)

Metoder

```
int countBulls(int guess, int answer) {
   int bulls = 0;
   int gDigit;
   int aDigit;
   while (guess > 0) {
       gDigit = guess % 10;
       aDigit = answer % 10;
       if (gDigit == aDigit) {
           bulls++;
       }
       guess = guess / 10;
       answer = answer / 10;
   }
   return bulls;
}

int countCowsAndBulls(int guess, int answer) {
   int cows_and_bulls = 0;
   int gDigit;
   while (guess > 0) {
       gDigit = guess % 10;
       if (numberContains(answer, gDigit)) {
           cows_and_bulls++;
       }
       guess = guess / 10;
   }
   return cows_and_bulls;
}

int digitAt(int number, int index) {
   final int N_DIGITS = 4;
   int digit = 0;
   int i = N_DIGITS - index;
   while (i > 0) {
       digit = number % 10;
       number = number / 10;
       i--;
   }
   return digit;
}
```

```
boolean hasUniqueDigits(int number) {
   int d0 = digitAt(number, 0);
   int d1 = digitAt(number, 1);
   int d2 = digitAt(number, 2);
   int d3 = digitAt(number, 3);
   return !(d0 == d1 || d0 == d2 || d0 == d3 ||
                   d1 == d2 || d1 == d3 || d2 == d3);
}

boolean numberContains(int number, int digit) {
   int nDigit;
   while (number > 0) {
       nDigit = number % 10;
       if (nDigit == digit) {
           return true;
       }
       number = number / 10;
   }
   return false;
}

// Brute force ...
int getNumber() {
   int min = 1234;
   int max = 8642;
   int n = -1;
   while (!hasUniqueDigits(n) || numberContains(n, 0)) {
       n = rand.nextInt(max) + min;
   }
   return n;
}
```

Tester av Metoder

```
void test() {
   out.println(digitAt(12345, 0) == 1);
   out.println(digitAt(12345, 2) == 3);
   out.println(digitAt(12345, 4) == 5);

   out.println(hasUniqueDigits(123));
   out.println(!hasUniqueDigits(1134));
   out.println(hasUniqueDigits(12534));

   out.println("Number " + getNumber()); // Inspect output
   out.println("Number " + getNumber());

   exit(0);
}
```

Man testar bara metoder som har med programlogiken att göra. In och utmatning testas inte.

Sammanfattning

- Man kan maska ut enskilda siffror ur heltal , använd % och /
- För att behärska komplexa problem används funktionell abstraktion och funktionell nedbrytning
- En metod är en avgränsad del av ett program som utför en specifik uppgift
- Metoder måste deklarera och kan därefter anropas
- Då man vid funktionell nedbrytning skapar metoder måste dessa testas!