Repetition och sammanfattning

Förelasning 9.1

TDA540 – Objektorienterad Programmering

CHALMERS

Felsökning

Ofta sker felsökning m.h.a. utskrifter av värden

- Ofta ok men…
- … till slut kan det bli så mycket utskrifter att man inte hittar
- När man är klar måste man ta bort allt (jobbigt)

Bättre: Använd en debugger (avlusare)

- En **debugger** är ett program som kör ditt program sats för sats (rad för rad)
- Efter varje sats pausar debuggern, man kan då inspektera variabler m.m.
- Efter pausen stegar man vidare (kör nästa sats genom att klicka)
- Man kan köra hela satser eller om det är metodanrop stega in i metoden och köra denna på samma sätt (kan stega ur också)
- För att få en första paus måste man ange en **brytpunkt**
- IntelliJ har en mycket bra inbyggd debugger

Typer

En **typ** (datatyp) anger vilken “sort” ett värde har

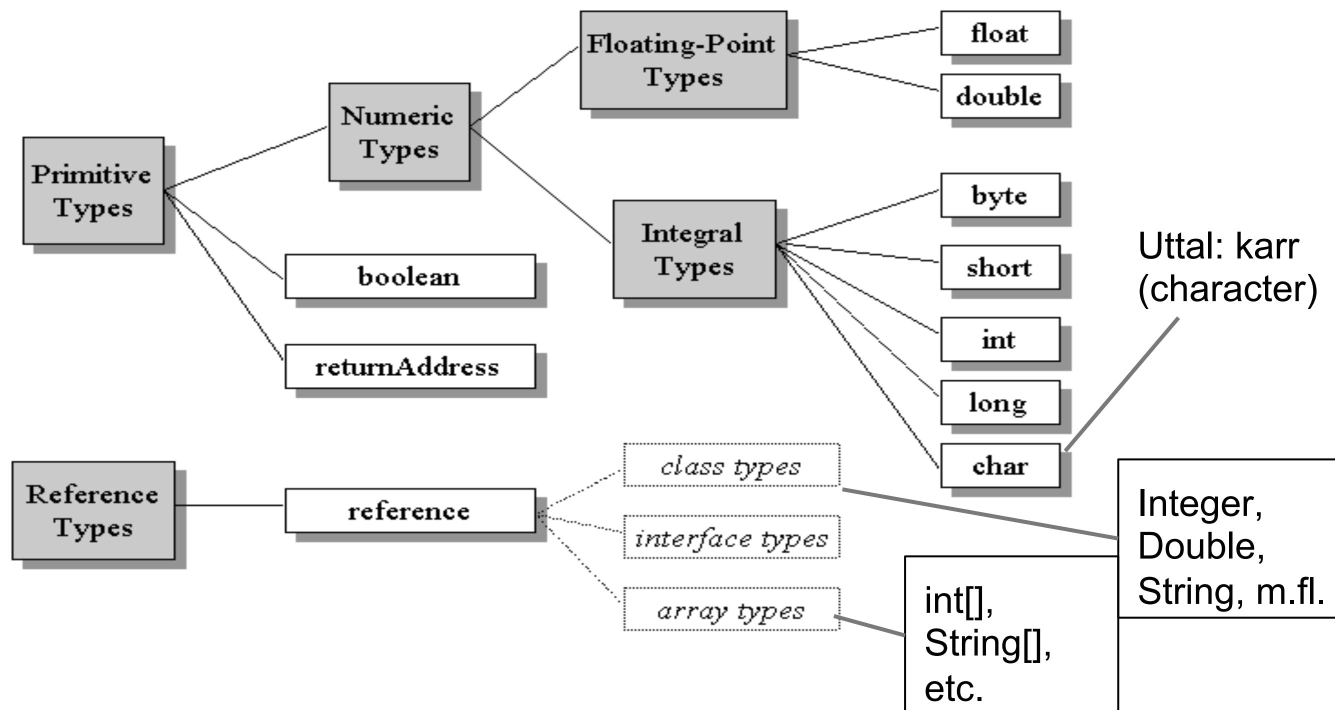
- Vilken mängd värdet ingår i (heltal, personer, bilar, …)
- Typer används för att inte blanda ihop olika sorters data
- Typen anger möjliga operationer för värdena
- Vissa operationer är orimliga för vissa typer (men inte för andra)
- m.m.

```
// Useless, div operation for String!
“olle” / 5       // Type error

// Ok!
5.5 / 5
```

Typer i Java

Alla värden i Java måste tillhöra någon av typerna nedan (vita rektanglar)



Typsystem

Ett typsystem kontrollerar att inga typfel finns i ett program

- Eftersom alla värden i Java har typer kan Javas typsystem (i kompilatorn) kontrollera att vi inte har några typfel i programmet redan vid kompileringen!
- Vi slipper fel under körning (sparar tid)!
- Typer och typsystem är tänkt som en hjälp för oss (men som nybörjare kan det upplevas annorlunda …)

Literaler

Literaler är “bokstavliga” värden skrivna direkt i koden (hårdkodade, kan inte ändras, de är “det de är”)

- Literaler tilldelas automatiskt en typ

```
// Java literals and types
1           // an int
1L          // a long (big integer)
1.0         // a double
1.0f        // a float
“1”         // a string
‘1’         // a char
true        // a boolean (only two values, the other is false)
null                // the null type (only one value, null)
```

Deklarationer

En deklaration berättar att något existerar, vad det är, vilket namn det har och vilka typer som är inblandade

- I Java måste "allt" som används i programmet deklareras; klasser, instansvariabler, metoder, variabler, ….datorn måste entydigt veta vad vi syftar på och vilka typer eventuella värden har

Synlighetsområde

Synlighetsområdet (**scope**) är den del i programmet där en deklaration gäller (där vi kan använda det deklarerade namnet)

- Gör att vi kan använda samma namn på olika ställen utan att tvetydigheter uppstår
- Vanligaste synlighetsområdet är mellan ett par krullparenteser "{" … "}" ett block, Java har block scope
- Om blocken är nästlade "{".. "{" … "}"... "}" kan det inre området komma åt namn i det yttre men inte tvärtom
- Alla namn måste vara unika inom sina respektive synlighetsområden
- Samma namn i inre område döljer namn i yttre

Variabler

Variabler är ett sätt att komma åt minnet

- Ett namn på en plats i minnet. Allt programmet skall komma ihåg måste sparas i variabler
- Alla variabler måste ha en typ. Typen anger vilka värden som kan lagras i eller refereras av variabeln
- Typen anges vid deklarationen av variabeln tillsammans med namnet m.m.
- Deklaration av variabler ger bara en variabel
- Variabler har en livslängd (normal då koden mellan “{“ och “}” körs)

```
// Declaring av variable
Integer i;  // Just the variable nothing else!
```

Primitiva- och Referensvariabler

En primitiv variabel har en primitiv typ. En referens-variabel har en referenstyp

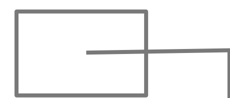
int i = 4;

Integer j = 4;

Namnlöst objekt

int är en primitiv typ för heltal.
I en primitiv variabel finns värdet "i variabeln"

Integer är en referenstyp för heltal
I en referensvariabel finns en referens till ett namnlöst "objekt" som innehåller värdet eller **null** (objekt saknas)



Integer k;

Debug av referens i IntelliJ visar ungefär java.lang.Integer@442"4" eller null

Operatorer

Operatorer beter sig som funktioner ...

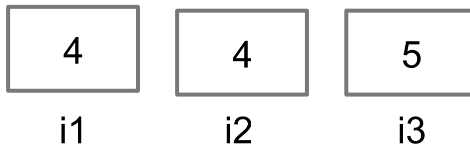
- … men skrivs på ett annat sätt (annan **syntax**)
- Aritmetiska; +, -, *. /, %
- Relations och logiska: !, !=, ==, > ,<, >= ,<= &&, ||,
- Tilldelnings operatorn: =
- Operatorer har prioritet och associerar till höger eller vänster
- Operatorer kräver vissa typer för operanderna (vissa fungera för blandade typer)

```
// Operators and types
2 + 2.5              // Ok!
true == "true"       // Hmm...
```

Likhet

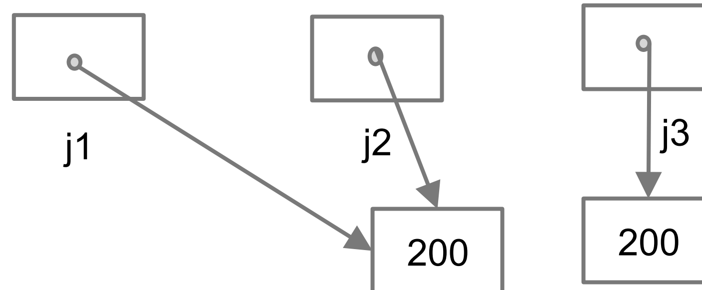
Likhet med primitiva- och referens-variabler

int i1 = 4;   int i2= 4;   int i3= 5;




```
i1 == i2    // True
i2 == i3   // False
```

Integer j1 = 200;   Integer j2 = j1;   Integer j3 = 200;




```
j1 == j2   // True
j2 == j3   // False
j2.equals(j3)  // True
```

Typkompabilitet

Om en typ är utbytbara mot en annan (inte leder till exekveringfel) är den **typkompatibel**

- I stället för typen heltal skulle man kunna använda typen reella tal (alla heltal kan ses som reella tal, de ingår i de reella talen).
- Tvärtom går inte!
- Finns väldigt många regler för typkompabilitet i Java

```
// Type compatibility
double x = 1;   // Different types but no problem!
```

Typomvandling

Om typer är kompatibla kan det vid behov ske **implicita** typomvandlingar (anpassning av typer)

```
// Type conversion
2 + 2.5      // Implicit conversion 2 -> 2.0, result
double
"2" + 3      // Implicit conversion 3 -> "3", result
String

// Boxing conversions
int i = new Integer(987);    // Ok, reference to primitive
Integer j = 123;             // Ok, primitive to reference
```

Explicit Typomvandling

Ibland måste vi uttryckligen omvandla en typ till en annan

Vanligt heltal <-> sträng och heltal <-> reellt tal

- int i = Integer.valueOf(“123”);
- String s = String.valueOf(123);

Uttryck

Uttryck byggs upp med värden, variabler och operatorer och representerar ett (enda) värde

- "An **expression** is a construct made up of variables, operators, and method invocations that evaluates to a single value"
- Uttryck kan inte skrivas fristående i programmet måste ingå i en sats (kommer snart)

```
// An expressions (can't write this alone in program)
Math.sqrt(4) + 2    // Single value is 4.0
```

Uttryck och Typ

Givet: Alla värden måste ha en typ
Givet: Ett uttryck representerar ett värde

Slutsats: Ett uttryck har en typ

```
// Some expressions and types
(x*x + y*y > 25) && (x > 0)     // Type?

sum/Math.sqrt(3.0 * 1.234)      // Type?
```

Uttryck med sidoeffekter

Sidoeffekt = Förutom att vi får ett värde "händer något" (minnet ändras)

```
// x++ and ++x are both expressions (represent values)
x++        // Sideffect: Increment (add 1 to variable x)
++x        // Also increment, … what's the difference?

// Difference is ...
x = 1;
y = x++;  // y will get value  1, value before incrementing
x = 1;
z = ++x;  // z will get value 2, value after incrementing

// 5++ , ++5  No! Must be a variable not a literal!
```

Paus

15 min

Sats

En **sats** är den minsta fristående enheten i Java (och andra imperativa språk). Representerar inte ett värde

- "A **statement** forms a complete unit of execution"
- En sats är ett imperativ (därav **imperativ programmering**)
- Avslutas med ";" (dock inte **styrande satser** som if, for, while, …)
- Ett program är en följd av satser (ungefär som meningar i en vanligt text).

```
// A statement
System.out.println("System.out.println");
```

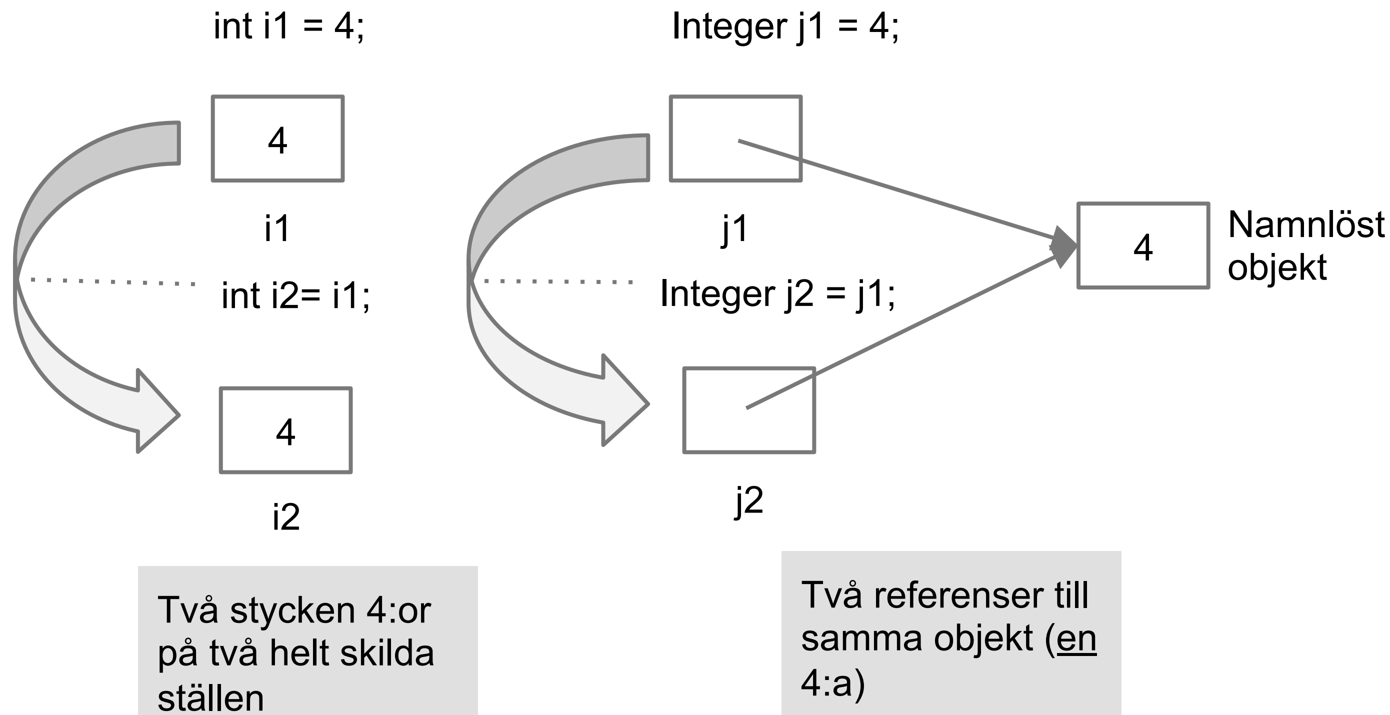
Tilldelning

En mycket vanlig sats är tilldelningssatsen

- Skrivs: höger sida = vänster sida
- Högersida är alltid en enda variabel
- Vänstersida är ett värde eller ett uttryck
- Om vänstersida ett uttryck så beräknas först värdet av detta därefter ...
- Kopieras vänstersida (värdet) till höger sida (lägg i variabeln)
- Höger sida måste vara typkompatibel med vänster

Tilldelning, forts.

Tilldelning med primitiva- och referens-variabler

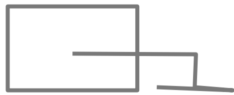


Fält (array)

En array håller ett fixt antal variabler av samma typ i en ordnad linjär struktur

- Typ för och antal variabler anges vid deklarationen
- Variablerna saknar namn, man kommer åt dem med namnet på fältet plus ett index (numret på variabeln, första har nummer 0)
- Man måste själv hålla reda på att 0 <= index < array.length
- En array är ett objekt, måste använda referensvariabler
- En deklartion ger inget array, måste skapa array:en
- Finns inga array-literaler men en förenklad syntax att skapa och initiera arrayer
- Kan avläsa storleken med array.length

Fält, forts.

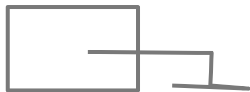


int[] iarr

Deklaration ger bara variabel

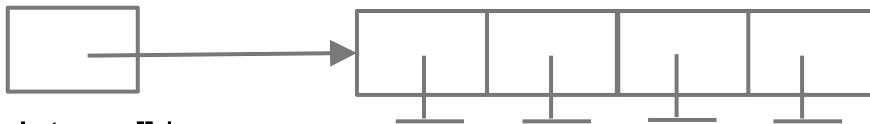
int[] iarr = { 4, 2, 0, 56 }

Deklaration och initiering



Integer[] jarr

Deklaration ger bara variabel



Integer[] jarr =
new
Integer[4];

Variablerna refererar inget!

Styrande Satser

För att styra i vilken ordning satser körs (exekveras) finns speciella konstruktioner s.k. styrande satser

- I princip finns bara 2 olika: Val (selektion) väljer mellan olika satser och upprepning (iteration), upprepar ett antal satser
- Av bekvämlighetsskäl finns flera varianter av varje:, if, if-else, if-else if-else, switch, for, while, do-while
- for används då vi vet hur många gånger vi vill upprepa
- while används då vi inte vet detta

Switch

En kompakt version av if-satsen

- Jämför endast på likhet
- Räknar upp ett antal “matchande fall”, case- grenar
- Matchar inget körs default-”grenen”
- Kan bara användas för heltal och strängar (bygger på equals() för strängar, mer senare)
- Viktigt med en break-sats i varje gren, annars körs flera grenar

```
// Switch statement
switch ( variableToTest ) {
    case value1:
      ... do something ...;
      break;
    case value2:
      ...do something ...;
      break;
    default:
      values_not_caught_above;
}
```

Kort for-loop

En kompakt version av for

- Då man inte har användning av något index (slipper index-fel)
- Typiskt läsa av elemeten i en array (eller samling, mer senare …)

```
// Short for-loop
int[] is = {1,2,3,4,5};
for (int i : is) {
	System.out.print( i );
}
```

Nästlade Styrande Satser

Mycket vanligt med nästlade styrande satser

- for-loop i foor-loop behövs då man har 2 dimensionella problem (rader, kolumner)
- if-sats i while/for-loop behövs då man skall välja något av ett antal värden i t.ex. en array

Metoder

En metod utför något med/på ett visst objekt

- Primitiva variabler har inga metoder
- Metoder kan fungera som uttryck (ha ett returvärde) eller satser (sakna returvärde)
- En metod anropas (inte “kallas på”)

```
int i = 4;
 i. …          // No! no methods, i not an object
Integer i = 4;
i.toString() // i an object, has methods
```

Punktoperatorn

För att anropa en metod på ett objekt används "."-operatorn på <u>referensen till objektet</u>

- Tolkas som: Följ referensen till objektet, anropa metoden på objektet

Metodanrop

Vid ett metodanrop sker ett hopp i programmet (till första raden i metoden)

- När metoden är klar sker ett återhopp till stället där anropet skedde (påbörjad sats).Kan ske i många led a() anropar b(), anropar c() … återhopp till b(), återhopp till a()
- Vid anrop kopieras aktuella parametrar till motsvarande formella (utifrån position). Kallas **värdeanrop (call by value)**
- Om parametrarna är ett uttryck beräknas först värdet av uttrycket därefter sker kopiering
- Typen på uttrycket måste stämma med typen på formella parametern (eller kompatibelt)
- Alla variabler i en metod (även parametrar) existerar bara då metoden körs (lokala variabler)!
- Om metoden returnerar ett resultat kopieras detta, vid tilldelning, till variablen

Metodanrop och Namn

I samband med metodanrop

Olika namn kan referera samma sak!

- int[] x i main kan referera samma som int[] y en metod (parameter)

Samma namn kan beteckna olika saker

- int x i main och int x i en metod är olika variabler

Strängar

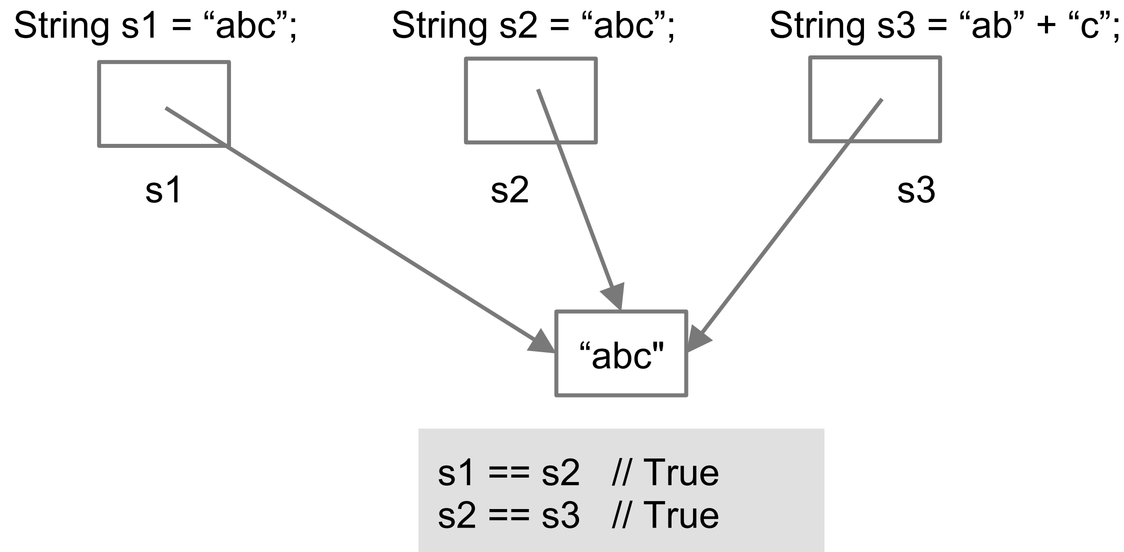
Används för godtyckligt långa texter (max runt 65000 tecken)

- Finns som objektliteral, “123”, en sträng är ett objekt
- Kan tilldelas till en referensvariabel av typen String
- Kan inte ändras, vid förändring skapas alltid nya strängar (objekt).
- Sammanslagning (konkatenering) av strängar (eller sträng och godtycklig typ) sker med +-operatorn. En ny sträng skapas. Bokstäverna kopieras från operander till den nya strängen
- Har equals() metod för teckenvis jämförelse, tecken för tecken från vänster till höger ...
- … och många fler användbara metoder.

Likhet för Strängar

Normalt används equals() för att jämföra strängar

- OBS, strängliteraler finns bara i en enda upplaga (alla referenser pekar på samma, jmf Integer)



Metoder för Strängobjekt

Några användbara metoder

```
String s = “abcdea”;
boolean b = s.equals(“abcdea”) // true
int i = s.indexOf(“a”);       // 0
int j = s.lastIndexof(“a”)    // 5
char ch = s.charAt(2)         // ‘c’
String sub = s.subString(3);  // “abcd”
String[] strs = s.split(“c”); // [ “ab”, “dea”]
String[] strs = s.split(“”);  // [ “a”,“b”,“c”,“d”,“e”,“a”]
String s2 = s.replace(“a”, “x”); // “xbcdea”
String s3 = s.replaceAll(“a”, “x”) // “xbcdex”
char[] chs = s.toCharArray();  // [‘a’,‘b’,‘c’,‘d’,’e’,‘a’]
```

System.in och out

Alla Java program tilldelas automatisk två “strömmar” när det startar

- System.in är en ström som man kan använda för att skicka data till programmet. Används tillsammans med t.ex. en Scanner
- Då man anropar metoder på Scanner blockeras programmet, det väntar tills man tryckt enter. Ett **blockerande anrop**!
- System.out är en ström som kan skicka data från programmet (till skärmen m.h.a. print() eller println()
- Det ovan sammanfattas till IO (inout/output)

```
// IO
Scanner sc = new Scanner(System.in);
String s = sc.nextLine();       // Always use nextLine()!
System.out.println(“bla bla bla”);
```

En Kommandoradsmeny

En kommandoradsmeny (command line menu) är ett äldre sätt att styra ett program

- En kommandoradsmeny är helt textbaserad (ingen grafik)
- Användaren väljer vad som skall göras genom att skriva ett kommande (en sträng) och därefter enter
- Programmet utför kommandot och skriver ut ev. utdata till skärmen (i textform)
- Kommandoradsmenyer har för det mesta ersatts av grafiska användargränssnitt (med knappar o.s.v. graphical user interfaces = GUI).
- En kommandorad är lättare att programmera därför använder vi inledningsvis en sådan

Slumptal

java.util.Random: "An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.)"

```
// Getting a "random" number
Random rand = new Random();
int rand = rand.nextInt(6);    // 0-5
```