مقدار دهی اولیه و پاک سازی

مقدمه

- با پیشرفت و توسعه کامپیوتر،شیوه برنامه نویسی ناامن یکی از عوامل اصلی در سخت بودن برنامه نویسی بود.
- بسیاری از باگ های مربوط به c زمانی اتفاق می افتند که برنامه نویس،مقدار دهی اولیه را فراموش می کند.
- پاک سازی(cleanup) نیز یکی دیگر از مشکلات موجود در برنامه نویسی است.زمانی که در رابطه به المانی،عملیات مد نظر را انجام دادیم، به راحتی آن المان را فراموش می کنیم .
- به همین دلیل جاوا از مفهوم constructor استفاده می کند.هنگامی که یک object ایجاد می شود،متدی خاص به صورت خودکار فراخوانی می شود.به علاوه جاوا با استفاده ازgarbage collector ، به صورت خودکار منابع حافظه را آزاد می کند.

مقداردهی اولیه تضمین شده با استفاده از constructor

- در زبان جاوا طراح کلاس با استفاده از constructor می تواند مقداردهی اولیه را تضمین کند.
- در صورتی که کلاسی دارای یک constructor باشد،جاوا هنگامی که objectای ایجاد شد،قبل از دستکاری آن توسط کاربر،به صورت خودکار آن constructor را فراخوانی می کند.
- نام constructor مشابه نام کلاس است

```
class Rock{
    Rock() {
        System.out.print("Rock ");
    }
}
```

مقداردهی اولیه تضمین شده با استفاده از constructor

- Constructor ،ای که هیچ ورودی ای نمی گیردconstructor پیش فرض نامیده می شود و در مستندات جاوا معمولا از اصطلاحno-arg constructor. برای آن استفاده می شود
- ورودی های Constructor روشی را برای ایجاد پارامتر جهت مقداردهی اولیه به یک object ،فراهم می کند.

```
class Rock{
    Rock(int i) {
        System.out.print( "Rock "+ i);
    }
}

new Rock2(12);
```

متد overloading

- از طریق نام می توان به objectها و متدها مراجعه کرد.گاهی اوقات یک نام میتواند معانی متفاوتی داشته باشد و این مفهوم overloading است.
- در بسیاری از زبان های برنامه نویسی(مخصوصا C) باید برای هر متد از یک شناسه واحد استفاده کرد و به عنوان مثال نمی توانیم تابعی به نام print( ) را برای چاپ اعداد صحیح استفاده نموده و تابعی دیگری نیز به همین نام داشته باشیم و از آن برای چاپ اعداد اعشاری استفاده کنیم.
- فرض کنید که کلاسی ایجاد کرده ایم که خودش به روشی استاندارد از طریق خواندن اطلاعات از یک فایل،مقداردهی اولیه را انجام می دهد.شما به دو نوع constructor نیاز دارید: constructor پیش فرض و constructor ای که یک رشته را به عنوان ورودی می گیرد.

چون هر دوی اینها constructor هستند باید هم نام با کلاس مد نظر باشند.بنابرین برای ایجاد چند متد با نام یکسان و ورودی های متفاوت استفاده از مفهوم overloading ضروری است.

متدoverloading

```
 Tree() {
    print("Planting a seedling");
     height = 0;
  }

  Tree(int initialHeight) {
     height = initialHeight;
     print("Creating new Tree that is " + height + " feet tall");
}
```

تعیین متدهای overloading

هر کدام از متدهای موجود در overloading باید لیست متفاوتی از آرگومان ها را با نوع داده های مشخص داشته باشند.

حتی تفاوت در ترتیب آرگومان ها برای تشخیص دو متد کافی است.

overloadingو نوع داده های اولیه

یک نوع داده اولیه می تواند به صورت خودکار از یک نوع کوچکتر به نوع داده بزرگتر بسط داده شود و این موضوع در ترکیب با مفهوم overloading کمی گیج کننده است.

overloading در مقدار خروجی

- دو متد زیر که هم نام و هم ورودی های یکسان دارند به راحتی از هم قابل تشخیص هستند:

```
void f() {}
int f() { return 1; }
```

اما زبان جاوا چگونه تصمیم بگیرد که کدام یک از این دو متد را فراخوانی کند؟و یا فردی که کدها را میخواند از کجا متوجه شود که کدام یک از متدها استفاده شده است؟

به خاطر این مشکلات و مشکلاتی دیگر ما نمی توانیم برای overloading متدها تنها از نوع داده های خروجی متفاوت استفاده کنیم.

Constructor پیش فرض

- Constructor پیش فرضconstructor ای است که هیچ آرگومانی را به عنوان ورودی نمی گیرد و برای ایجادobject پیش فرض استفاده می شود. در صورتی که کلاسی هیچconstructor ای نداشته باشد،کامپایلر جاوا به صورت خودکار یکconstructor پیش فرض ایجاد می کند و در صورتی که در کلاسconstructor ای را تعریف کنید (چه با آرگومان های ورودی و چه بدون ورودی) ،کامپایر دیگر ازconstructor. پیش فرض استفاده نمی کند

```
class Bird2 {
  Bird2(int i) {}
  Bird2(double d) {}}
public static void main(String[] args) {
    //! Bird2 b = new Bird2(); // No default
    Bird2 b2 = new Bird2(1);
    Bird2 b3 = new Bird2(1.0);
  }
}
```

کلیدواژه this

- فرض کنید که دو object از یک نوع داده داریم.مثل a و b
- در صورتی که بخواهیم متدی مانند متدpeel( ) را برای هر دوی این object ها فراخوانی کنیم به صورت زیر عمل می کنیم:

```
public class BananaPeel {
  public static void main(String[] args) {
   Banana a = new Banana(),  b = new Banana();
   a.peel(1);
   b.peel(2);
  }
 }
```

کلیدواژه this

- کامپایر جاوا برای آنکه کاربر بتواند کد نویسی را به شیوه شی گرایی انجام دهد و از طریق آن پیغامی را به یک object ارسال نماید،یک سری عملیات مخفی انجام می دهد.
- اولین آرگومان مخفی برای متد peel() فرستاده می شود و این آرگومان به عنوان رفرنسی به object ای که دستکاری شده خواهد بود.
- برای انجام این کار می توان از کلید واژه this استفاده نمود.از این کلید واژه تنها می توان داخل متدهای غیر استاتیک استفاده کرد و referenceای را برای objectای که متد برای آن فراخوانی شده،ایجاد می کند.
- توجه داشته باشید که در صورتی که متدی از کلاس مد نظرتان را در طول متد دیگری از کلاس فراخوانی کنید،نیازی به تعریف this نمیباشد.

فراخوانی یک constructor در constructorای دیگر

- هنگامی که چندین constructor برای یک کلاس دارید،در مواردی ممکن است تمایل داشته باشیم تا constructorای را در constructor دیگر فراخوانی کرده و از این طریق کدنویسی را کوتاه تر کنیم.این فراخوانی می تواند از طریق کلید واژه this صورت گیرد.
- در یک constructor،کلید واژه this ، یک فراخوانی صریح را به constructor ای که آرگومان های ورودی اش با ورودی ها هماهنگ است ،انجام می دهد.

```
public class Flower {  int petalCount = 0; String s = "initial value";
    Flower(int petals) {
        petalCount = petals; }
    Flower(String ss) {
        s = ss;   }
    Flower(String s, int petals) {
        this(petals);   //!    this(s); // Can't call two!
        this.s = s; }
    Flower() {
        this("hi", 47);
}
}
```

مفهوم Static

- برای متدهای static کلید واژه this وجود ندارد و نمی توان یک متد غیر استاتیک را در داخل یک متد استاتیک فراخوانی کرد.
- این کار مانند زمانی است که معادلی از یک متد global را ایجاد می کنیم.
- برخی بر این باورند که استفاده از متدهای استاتیک خلاف شی گرایی است چرا که آن ها قواعد متدهای global را دارند.

پاکسازی:استفاده از finalize و garbage collector

- زبان برنامه نویسی جاوا از garbage collector برای آزاد سازی حافظه objectهایی استفاده می کند که دیگر در برنامه استفاده نمی شوند.
- البته یک مورد استثنا نیز وجود دارد:فرض کنید object ای بدون استفاده از new به نقطه مشخصی از حافظه اختصاص داده شده باشد. garbage collector تنها حافظه ی objectهایی را آزاد می کند که از راه new ایجاد شده اند و نمی تواند نقاط مشخص حافظه را شناسایی کند.برای حل این مشکل می توانیم متد finalize( ) را در کلاس مد نظر خود ایجاد کنیم.

پاکسازی:استفاده از finalize و collector garbage

- برخی برنامه نویسان ممکن است در ابتدا متد **finalize( )** را با destructor موجود در c++ اشتباه بگیرند. destructor تابعی است که زمانی که یک object از بین میرود می شود استفاده می شود.

1) Object ها ممکن است هیچ گاه توسطcollector garbageاز بین نروند .

2) collector garbage.یک فرایند تخریب نیست

مقداردهی اولیه اعضا

- جاوا از روشی استفاده می کند که مقدار دهی اولیه متغیرها را پیش از استفاده از آن ها، به شیوه مناسب تضمین کند.
- یکی از روش های مستقیم در تعیین مقداردهی اولیه این است که مقدار را در نقطه از کلاس که متغیر در آن تعریف شده است،تخصیص دهیم.
- از این روش می توان برای مقداردهی اولیه object های غیر اولیه (non-primitive) نیز استفاده کرد.
- همچنین برای مشخص کردن مقدار در مقداردهی اولیه می توان یک متد را فراخوانی کرد.

```
public class InitialValues2 {
    boolean bool = true;
    char ch = 'x';
    byte b = 47;
    int[] a1 = { 1, 2, 3, 4, 5 };
    Depth d = new Depth();
    int i = f();
  }
```

مقداردهی اولیه در Constructor

- می توان از Constructor ها نیز جهت مقداردهی اولیه استفاده نمود و این قابلیت باعث ایجاد انعطاف بیشتر در برنامه نویسی شده است؛چرا که می توان متدها را فراخوانی کرد و عملیات مد نظر جهت مقداردهی اولیه را در زمان اجرا انجام داد.

```
public class Counter {
  int i;
  Counter() { i = 7; }
  // ...
}
```

- شیوه و ترتیب مقداردهی اولیه: قواعد مقداردهی اولیه به وسیله شیوه و ترتیبی که متغیرها در داخل کلاس تعریف شده اند،تعیین می شود.

مقداردهی static به صورت صریح

- جاوا این اجازه را به کاربر می دهد که مقداردهی اولیه به متغیرهای static را به صورت گروهی در داخل بلاک های استاتیک (static block) در کلاس انجام دهد.مانند زیر:

```
public class Spoon {
   static int i;
   static {
     i = 47;
   }
}
```

مقدار دهی اولیه متغیر محلی غیر استاتیک

- جاوا از قواعد کدنویسی مشابه ای نیز برای مقداردهی اولیه متغیرهای غیر استاتیک هر object استفاده می کند که به آن instance initialization می گویند.

```
public class Mugs {
  Mug mug1;
  Mug mug2;
 {
    mug1 = new Mug(1);
    mug2 = new Mug(2);
    print("mug1 & mug2 initialized");
  }
}
```

لیست آرگومان های متغیر Varargs

```
static void printArray(Object[] args) {
     for(Object obj : args)
      System.out.print(obj + " ");
}
```

- در ورژن های قبل از Java SE5 برای ایجاد لیست آرگومان های متغیر از کدی مشابه بالا استفاده می شد.در Java SE5 خصوصیت امکان درخواست های طولانی اضافه شد و می توان از چند نقطه جهت تعریف لیست آرگومان متغیر مانند آنچه در printArray( ) می بینید استفاده کرد.

```
static void printArray(Object... args) {
   for(Object obj : args)
        System.out.print(obj + " ");
}
```

لیست آرگومان های متغیر Varargs

- با استفاده از لیست آرگومان های متغیر(vararg‌ها) دیگر نیازی نیست که به طور کامل کد یک آرایه را بنویسیم و کامپایلر وقتی لیست آرگومان های متغیررا تعیین کنید به درستی آن را پر می کند.

```
static void f(int required, String... trailing) {
    System.out.print("required: " + required + " ");
    for(String s : trailing)
        System.out.print(s + " ");
}
  public static void main(String[] args) {
     f(1, "one");
     f(2, "two", "three");
     f(0);
  }
```

نوع شمارشی enum

- یکی از ویژگی های جدید اضافه شده به Java SE5 کلید واژه enum است که در مواردی که نیاز به گروهی از متغیرها داریم و میخواهیم مجموعه ای را به صورت فهرست وار ایجاد کنیم،استفاده می شود.
- از آنجا که switch جهت استفاده از مجموعه محدودی از انتخاب ها در نظر گرفته شده، در آن میتوان از enum ها نیز استفاده کرد .

```
public enum Spiciness {
  NOT, MILD, MEDIUM, HOT, FLAMING
}
public class SimpleEnumUse {
   public static void main(String[] args) {
     Spiciness howHot = Spiciness.MEDIUM;
     System.out.println(howHot);
  }
}
```