

Index

SYMBOL

`{}` (curly braces), 122

A

abstract class, 294–295

abstraction, 1

accessibility

- generic classes, 70–73
- protected members, 55–56
- static constructors, 51

accessing items

- Bag<T> library, 326–327
- BCL collections, 142–144
- BigList<T> class, 334–335
- double-ended queue, 339–341

action on object

- delegates, 106–107
- list, applying, 163

adding items

- BCL collections
 - dictionary, 179
 - internally implemented classes, 165–166
 - keyed items, 159
 - keys, mapping to values, 152–153
 - linked list, 160
 - overall, 136–141
 - queue, 175
 - sorted dictionary list, 182

Power Collections library

- Bag<T> class, 321
- Bag<T> class sorted version, 347–348

BigList<T> class, 332

CollectionBase<T> class, 336

double-ended queue, 338–339

Set<T>, 362

algorithms, Power Collections library

- class, 295
- comparers and comparisons, 305, 321
- contents, inspecting, 302–304
- copying and converting collections, 305–307
- lexicographical comparison, 307–309
- listed, 296–301
- modifying and arranging contents, 309–313
- randomizing collections, 321
- searching collections, 313–316
- set operations, 318–321
- sorting collections, 316–318

aliasing, 231–232

ambiguity

- mixing classes and interfaces, 124–125
- problems, generally, 123–124

anonymous methods, 109

API (Applications Programming Interface)

- generic collections, benefits of using, 218
- interface
 - choosing least specialized, BCL, 235–236
 - using instead of classes, 240–241
- reflection
 - inheritance, 198–199
 - methods, 199–201
 - parameters and arguments, examining, 195–198

arguments

reflection, 195–198

type

alternative template implementation, 43

constructed types, caution against, 232–233

described, 15

J#, 288

arity, type parameters, 17

arranging collection contents, 309–313

arrays

of collection, 301

`CollectionBase<T>`, 337

concatenating, 297, 312

J# generic types, 289–290

performance, 259–260

type parameters, 65–66

assemblies, template, 42–43

B

backward compatibility, .NET, 262

`Bag<T>` class

Power Collections library

accessing items, 326–327

construction and population, 324–325

removing items, 327–329

set operations, performing, 329–332

sorted version

adding and removing items, 347–348

described, 346–347

range operations, 348–349

base classes

behavior, generalizing with polymorphism, 13

inheritance

C++ generic classes, 266–267

generally, 52–55

generic constraints, 126–127

polymorphism, achieving, 13

protected members, 55–56

reflection, 198–199

type, specifying, 28

Power Collections library, 292–293

BCL (Base Class Library) generics

API interface, choosing least specialized, 235–236

benefits of using, 129

classes, 132–133

collections

accessing and inspecting, 142–144

adding and updating, 136–141

constructing, 135–136

described, 134

functions, extending, 144–147

removing, 141–142

components, list bound to various, 184

custom collections, 235

delegates, 134

dictionary

sorting based on key, 182–183

specific order, accessing, 179–182

enumerators, 133–134

equality, testing, 157–158, 184–185

“for each” iteration, enabling, 236

interfaces, 130–132

internally implemented classes

adding and updating items, 165–166

described, 162–163

removing items, 167–168

searching for items, 168–171

size, pre-allocating, 164–165

sorting items, 173–175

transforming contents, 171–173

keys, mapping to values

adding and updating items, 152–153

constructors, 150–152

described, 149–150

looking up, 154–156

retrieving, 156–157

linked list

creating, 160–161

type-safe wrapper, 161

null collection elements, 185

Power Collections framework and, 291

protected properties and methods, 159–160

queue, 175–177

read-only collection, 177–179

stacked class, 183–184

`System.Collections.ObjectModel`

namespace, 133

binary search

`BigList<T>`, 332

Power Collections algorithm, 296

`BinaryPredicate<T>` delegate, 336

bloat

function, allowing varying data, 84

non-generics method, 7

templates, 42

blurring lines, C++, 264

boxing, rejection of

- .NET, 245
- numbers to use as `object`, 5
- performance issues, 259–260

braces, curly ({}), 122**C****C# programming language**

- accessibility rules, 72–73
- angle bracket syntax, 131
- anonymous methods, 109
- class constraints, 125
- delegates, supplying, 168
- method signature, 63
- multiple constraints, 122–123
- nullable types, declaring, 76
- overriding generic methods, 93, 94
- parameter names, 228
- ref types, using type parameters for, 220–221

C++ programming language. See also .NET

- blurring lines, 264
- code bloat, 42
- compatibility, 264
- constraints, 42, 271–273
- generic classes
 - consuming, 265–266
 - default types, 268
 - described, 264–265
 - inheritance, 266–267
 - methods, 268
 - nested, 267–268
- generic delegates, 273–274
- generic interfaces, 270
- generic methods, 268–270
- STL.NET, 277–278
- templates
 - generics versus, 263
 - least common denominator programming, 20
 - mixing with generics, 274–277

cache, 59**capacity**

- list
 - pre-allocating, 164–165
 - setting, 163
- queue, 176
- stack, 184

case sensitivity, comparisons, 147**casting**

- consequences of using, 29–31
- constraint, absence of, 121
- exact run-time types, 255–256

checking type parameters, 27–28**clarity, type safety versus, 32–33****class**

- BCL generics
 - generally, 132–133
 - `List<T>` class, 162
- generic constraints, 116–117
- generic type, appending, 9
- interfaces versus, 240–241
- type parameter, descended from, 44

client

- calling code, remote access, 213–215
- type-safe interface, 223

closed constructed types, 15–16**closed types**

- constructed fields, 59–60
- converting to open, 194–195
- creating, 189–193
- reflection, 188

code bloat

- function, allowing varying data, 84
- non-generics method, 7
- templates, 42

code sharing

- just-in-time specialization, 250–251
- .NET
 - code sharing, 248
 - code specialization, 248
 - hybrid model, 248
 - just-in-time, 249–253
- NGen, 257

code specialization

- BCL custom classes, 235
- .NET
 - code sharing, 248
 - code specialization, 248
 - hybrid model, 248
 - just-in-time, 249–253
- templates, 43–44

CollectionBase<T> abstract class, 294–295**collections**

- BCL generics
 - accessing and inspecting, 142–144
 - adding and updating, 136–141

collections (continued)

- constructing, 135–136
 - described, 134
 - functions, extending, 144–147
 - removing, 141–142
 - clients, allowing to add or remove items, 131
 - editing
 - adding, 309–311
 - concatenating arrays 312
 - reversing order, 312
 - rotating, 312–313
 - dictionary versus, 152
 - enumerating content, 133–134
 - generics guidelines, 218
 - inspecting contents, 302–304
 - Power Collections library
 - implementing custom (`CollectionBase<T>` abstract class), 336–337
 - ordered view, 349–350
 - type-safe stack, 49
- Collection<T> class**
- accessing and inspecting items, 142–144
 - constructing, 135–136
 - described, 134–135
 - extending functions, 144–147
 - populating and maintaining state, 136–141
 - removing items, 141–142
- comparison**
- BCL objects, 147–149
 - Power Collections algorithms
 - `Bag<T>`, 322, 323
 - listed, 305–307
 - `Set<T>`, 362
 - `Triple` class, 366
- compatibility**
- backward with .NET, 262
 - C++, 264
- compiler**
- overloading methods, 62–63
 - type, generating, 43
 - type parameters, comparing blocked, 158
- compile-time instantiation, 36–37, 40, 43**
- complex types, aliasing, 231–232**
- components, BCL list bound to various, 184**
- concatenating arrays, 297, 312**
- consecutive values, searching for pattern of, 315**
- console application code, 212–213**
- constraints**
- C++, 271–273
 - delegates, applying, 108–109

- generic types, declaring, 40
 - hidden, avoiding, 237
 - J#, 285–287
 - least restrictive, choosing, 236–237
 - methods, 87–88, 91
 - multiple ambiguity, avoiding, 237–238
 - parameterless constructors, providing, 238–239
 - reflection, 197
 - type parameter names, qualifying, 229–230
- constructed fields, 59–60**
- constructed types**
- described, 15
 - freedoms, 69
- construction**
- `Bag<T>` library, 324–325
 - BCL collections, 135–136
 - large lists, managing (`BigList<T>` class), 333–334
- constructors**
- generic constraints, 118–119
 - keys, mapping to values, 150–152
- consuming, C++ generic classes, 265–266**
- container**
- `CollectionBase<T>` abstract class, 294–295
 - dictionary base classes, 295
 - generic, 4
 - `ListBase<T>` abstract class, 293–294
 - types, 133
- converting**
- `CollectionBase<T>` class, 337
 - customer names list, 173
 - dictionary items, 359
 - list to array of items, 163
 - objects to appropriate data type
 - consequences of using, 29–31
 - constraint, absence of, 121
 - exact run-time types, 255–256
 - Power Collections algorithms, 297, 305–307
 - type, 67
- copying**
- BCL collection
 - dictionary items, 179
 - linked list to `Array`, 160
 - list, 171–173
 - list contents to `Array`, 162
 - queue to `Array`, 175
 - read-only, 177
 - stack items, 183
 - Power Collections algorithms
 - `BigList<T>`, 332
 - `CollectionBase<T>`, 337

collections, 297, 305–307
 Set<T>, 362

counting

BCL collection
 capacity, 165
 dictionary, 179
 general collection items, 136
 keyed collection items, 159
 linked list nodes, 160
 list items, 162
 queue, 175
 read-only, 177
 sorted list, 182
 Power Collections library
 algorithm, 297
 Bag<T>, 323
 BigList<T>, 332
 dictionary, 359
 Set<T>, 362
 type parameters, 17

criteria, delegate, 107

cross-language support

generics, 44
 J#, 281–283

curly braces ({}), 122

custom

BCL collections, 235
 serialization, 204–208

customer

dictionary, creating, 180–182, 345–346
 domain object, building, 137–140
 interfaces preserving type safety, 25
 list
 adding to, 166
 converting, 173
 deleting from, 167
 read-only, 178
 sorting, 109, 173–175
 orders
 iterating and dumping out information, 24–25
 populating with, 23–24
 queue, 176–177
 type parameters, setting, 49

D

data collection

BCL generics, 130–132
 C++ class template declaration, 274–277
 classes versus, 240–241

constraints, 123–124
 generic classes, 81–82
 generic constraints, 115
 generics guidelines, 227
 J#
 basic, 283–285
 constraints, 285–287
 remote access, 211–212
 type parameters in, 62–63
 type safety
 described, 31
 preserving, 25–26

data container

CollectionBase<T> abstract class, 294–295
 dictionary base classes, 295
 generic, 4
 ListBase<T> abstract class, 293–294
 types, 133

data input

BCL generics, 130–132
 C++ class template declaration, 274–277
 classes versus, 240–241
 constraints, 123–124
 generic classes, 81–82
 generic constraints, 115
 generics guidelines, 227
 J#
 basic, 283–285
 constraints, 285–287
 remote access, 211–212
 type parameters in, 62–63
 type safety
 described, 31
 preserving, 25–26

data, populating

Bag<T> library, 324–325
 collection, adding, 136
 customer objects with orders, 23–24
 domain object, building, 137–140
 generic types, generally, 9–10
 large lists, managing (BigList<T> class), 333–334
 object class, non-generics programming, 5
 Set<T>, 364–365

data type

casting
 consequences of using, 29–31
 constraint, absence of, 121
 exact run-time types, 255–256
 varying only by, 221–223

debugging, 44

declaration, .NET validation, 244

default keyword, generic class, 73–74

default type, C++ generic class, 268

definitions

- generic type, 14–15
- open type, 193

delegates

- action on object, 106–107
- anonymous methods, 109
- BCL generics, 134
- choosing, 107
- constraints, applying, 108–109
- criteria, item meeting set, 107
- described, 99–102
- event handlers, 105–106
- generics and, 102–105
- methods, 96–97, 227
- with methods, 106
- Power Collections library, 296
- supplying, 168
- two objects, comparing, 107
- type coercion, 107–108
- type, converting, 107

deleting items

- algorithm, 313
- Bag<T> class
 - duplicates, 323
 - library, 327–329
 - method, 321
 - sorted version, 347–348
- BCL collections
 - dictionary, 179
 - generally, 141–142
 - keyed objects, 159
 - keys mapped to values, 157
 - list, 163
 - listed list nodes, 160
 - from queue, 175
 - sorted list, 182
 - stack, 183
- BigList<T>, 332, 333
- dictionary, 345
- double-ended queue, 339–341
- internally implemented classes, 167–168
- large lists, managing (BigList<T> class), 335–336
- object at specified index, 136
- read-only collection, 300
- Set<T>, 362, 363

derived dictionary types, 341

dictionary

- BCL generics
 - items, adding and editing, 179
 - keys, 180–181
 - sorting, 182–183
- collection, managing, 134
- interface, 132
- keys, mapping to values (Dictionary<TKey, TValue> class)
 - adding and updating items, 152–153
 - constructing, 150–152
 - described, 149–150
 - looking up items, 154–156
 - null keys, 157
 - removing items, 157
 - retrieving keys and values, 156–157
- Power Collections library
 - container base classes, 295
 - key, 298, 359–360
 - multiple values, accessing with key, 343–345
 - multiple values, extending, 345–346
 - states, locking, 361–362
 - subset, synchronized view of, 354

difference, comparing two collections

- algorithm, 301
- Bag<T>, 322
- Set<T>, 362, 363

discerning types, reflection, 188–189

disjoint collections, finding, 297, 363

domain objects

- associating with “child” objects, 3–4
- common attributes, holding, 2

double-ended queue

- accessing and removing items, 339–341
- adding items, 338–339
- described, 337–338

dumping

- customer order information, 24–25
- type information for generic classes, 76–77
- type parameters, 77–78

duplicates, removing from bag, 323

E

empty

- collection elements, 185
- keys, mapping to values, 157
- state, 241
- value, discerning, 74–76

end, adding to `BigList<T>` class, 332

enumerators

- BCL collection
 - dictionary, 179
 - generally, 133–134
 - linked list instance, 161
 - list, 163
 - queue, 176
 - read-only, 177
 - stack items, 183
- Power Collection
 - `Bag<T>`, 322
 - `BigList<T>`, 332
 - `CollectionBase<T>`, 337
 - read-only dictionaries, 361
 - `Set<T>`, 362

equality testing

- BCL generics, 157–158, 184–185
- Power Collections, 297
- queue objects, 175
- two objects, 132

events, generic classes, 77–80

events handler

- data interface, 241–242
- delegate, 105–106
- OOP, 30

exact run-time types, 254–256

exception handling

- data interface, 241–242
- delegate, 105–106
- OOP, 30

extending reflection, 188

eXtensible Markup Language (XML), 210

F

fields, generic class

- constructed, 59–60
- static, 57–59
- syntax, 56–57

filling entries, 313

filling with data

- `Bag<T>` library, 324–325
- collection
 - adding, 136–137
 - updating, 137–140
- data interface, 241–242
- domain object, building
- customer objects with orders, 23–24

generic types, generally, 9–10

- large lists, managing (`BigList<T>` class), 333–334
- object class, non-generics programming, 5
- `Set<T>`, 364–365

finding

- BCL collection
 - dictionary keys, 180
 - index of object, 136
 - keyed collection item, 159
 - linked list nodes, 160
 - list items, 136
 - methods, 170
 - read-only, 177
 - sorted list keys, 182
 - stack items, 183
- parameters, 47–48, 220
- Power collection
 - algorithm, 297
 - `CollectionBase<T>`, 337

for each loop

- BCL generics, enabling, 236
- casting errors, handling, 30

frequently used types, aliasing, 231–232

functions

- BCL collections, extending, 144–147
- bloat, 84
- values, comparing and returning greater, 83

G

generalization, 1

generic classes

- accessibility, 70–73
- C++
 - consuming, 265–266
 - default types, 268
 - described, 264–265
 - inheritance, 266–267
 - methods, 268
 - nested, 267–268
- default keyword, 73–74
- fields
 - constructed, 59–60
 - static, 57–59
 - syntax, 56–57
- indexers, properties, and events, 77–80
- inheritance
 - generally, 52–55
 - protected members, 55–56

generic classes (continued)

generic classes (continued)

- interfaces, 81–82
- just-in-time specialization, 249–250
- methods
 - arrays of type parameters, 65–66
 - described, 60–61
 - operator overloading, 66–67
 - overloading, 61–63
 - overriding, 63–65
- nested, 67–69
- null value, discerning, 74–76
- overloaded types, 50
- parameters, 47–48, 220
- static constructors, 51
- structs, 80–81

generic constraints

- class, 116–117
- constructor, 118–119
- described, 111–115
- drawbacks, 127–128
- inheritance, 126–127
- interface, 115
- multiple, 122–125
- types, 117–118
- value type parameter box, blocking, 119–122

generic delegates

- C++, 273–274
- described, 102–105
- J#, calling, 289
- methods, 227
- multiple, replacing, 225–227

generic inheritance, reflection, 198–199

generic interfaces, C++, 270

generic methods

- applying to individual, 16
- C++, 268–270
- just-in-time specialization, 252–253
- reflection, 199–201
- type inference versus, 233–234

generic structures, 295–296

generic types

- aliasing, 231–232
- arguments
 - caution against constructed types as, 232–233
 - described, 15
- defined, 17
- inference versus methods, 233–234
- info, accessing, 76–77
- instantiation, 16–17

- J#, consuming, 280–281
- open and closed, 15–16
- parameters
 - arity, 17
 - C# ref types, 220–221
 - described, 14–15, 49–50
 - names, 228–231
 - overuse, 233
 - `System.type`, replacing with, 219–220

generics

- benefits of using, 1–9, 12–13
- C# ref types, using type parameters for, 220–221
- collections, 218
- constructed types, 15
- cross-language support, 44
- data type, varying only by, 221–223
- debugging, 44
- defining, 217–218
- evolving nature of, 217
- “Hello Generics” example, 10–12
- interfaces, 227
- methods, applying to individual, 16
- objects, replacing with type parameters, 219
- organizing, 218
- parametric polymorphism, 13–14
- readability, balancing with expressiveness, 227–228
- refactoring, candidate methods for, 223–225
- run-time instantiation, 37–38, 42
- shell, building, 15
- static methods, 234
- templates versus, 35, 38–42

H

handler

- data interface, 241–242
- delegate, 105–106
- OO# 30

hash code, Triple class, 366

“Hello Generics” example, 10–12

hidden constraints, 237

hierarchies

- C++ generic classes, 266–267
- generally, 52–55
- generic constraints, 126–127
- polymorphism, achieving, 13
- protected members, 55–56
- reflection, 198–199
- type, specifying, 28

I**IComparer<T> interface, 132****identical members, constraints exposing, 123–124****IEnumerable<T> interface**

container support, 134

described, 130–131

“for each” iteration, enabling, 236

IEnumerator<T> interface, 132**IEqualityComparer<T> interface, 132, 299****IL**

code, examining, 38

.NET representation, 246–247

run-time instantiation, 37–38

index

collection object, finding, 136

key in sorted list, 182–183

keyed collection object, finding, 159

list items, 162, 163

Power Collection object

BigList<T>, 333

described, 297–298

pattern, searching for, 315–316

read-only dictionary, 361

read-only collection, 177

indexers

dictionaries, 153

generic classes, 77–80

Power Collection dictionary class, 341

sorted, 179

inference, type

generics guidelines, 233–234

methods, 95

inheritance

C++ generic classes, 266–267

generally, 52–55

generic constraints, 126–127

polymorphism, achieving, 13

protected members, 55–56

reflection, 198–199

type, specifying, 28

inserting

linked list node, 160

list items, 163

new object into collection, 136

range into BigList<T>, 333

inspecting BCL collections, 142–144**interface**

BCL generics, 130–132

C++ class template declaration, 274–277

classes versus, 240–241

constraints, 123–124

generic classes, 81–82

generic constraints, 115

generics guidelines, 227

J#

basic, 283–285

constraints, 285–287

remote access, 211–212

type parameters in, 62–63

type safety

described, 31

preserving, 25–26

internally implemented classes, BCL

adding and updating items, 165–166

described, 162–163

removing items, 167–168

searching for items, 168–171

size, pre-allocating, 164–165

sorting items, 173–175

transforming contents, 171–173

intersection points

in Bag<T>, 322, 331–332

in collection, 296

in Set<T>, 363

J**J# programming language**

arrays of generic types, 289–290

constraints, 285–287

cross-language support, 281–283

generic delegates, calling, 289

generic types, consuming, 280–281

interfaces, 283–285

limitations, 279

methods, calling, 287–288

migrating from Java generics, 280

type arguments, 288

JIT (just-in-time)

specialization

code sharing structures, 250–251

generic classes and shared code, 249–250

generic methods and shared code, 252–253

open types in shared code, 251–252

type instantiation, 16–17

K

keys

- BCL generics
 - adding and updating items, 152–153
 - constructors, 150–152
 - described, 149–150
 - looking up, 154–156
 - retrieving, 156–157
- dictionary, returning all, 180
- random access and ordered collection, 350–354

L

language, C# programming

- accessibility rules, 72–73
- angle bracket syntax, 131
- anonymous methods, 109
- class constraints, 125
- delegates, supplying, 168
- method signature, 63
- multiple constraints, 122–123
- nullable types, declaring, 76
- overriding generic methods, 93, 94
- parameter names, 228
- ref types, using type parameters for, 220–221

language, C++ programming. *See also* .NET

- blurring lines, 264
- code bloat, 42
- compatibility, 264
- constraints, 42, 271–273
- generic classes
 - consuming, 265–266
 - default types, 268
 - described, 264–265
 - inheritance, 266–267
 - methods, 268
 - nested, 267–268
- generic delegates, 273–274
- generic interfaces, 270
- generic methods, 268–270
- STL.NET, 277–278
- templates
 - generics versus, 263
 - least common denominator programming, 20
 - mixing with generics, 274–277

language, eXtensible Markup (XML), 210

language, J# programming

- arrays of genetic types, 289–290
- constraints, 285–287
- cross-language support, 281–283

- generic delegates, calling, 289
- generic types, consuming, 280–281
- interfaces, 283–285
- limitations, 279
- methods, calling, 287–288
- migrating from Java generics, 280
- type arguments, 288

large lists, managing (`BigList<T>` class)

- accessing contents, 334–335
- construction and population, 333–334
- described, 332–333
- removing items, 335–336

last item, returning

- linked list, 161
- list, 162

last-in, first-out list, 183–184

late-binding, creating types at run-time

- closed types, converting to open, 194–195
- creating open and closed types, 189–193
- described, 187
- discerning types, 188–189
- extending, 188
- generic inheritance, 198–199
- generic methods, 199–201
- obfuscated environments, 201–202
- open and closed types, 188
- parameters and arguments, 195–198

least common denominator programming

- object, managing, 3–4
- programming method, 20
- reducing reliance on, 219
- sorting objects of any type, 220

least restrictive constraints, 236–237

lexicographical comparison of collections, 307–309

libraries, 367. *See also* Power Collections library

linked list

- creating, 160–161
- type-safe wrapper, 161

list-based derived types, 342–343

`ListBase<T>` abstract class, 293–294

lists, read-only, 360–361

`List<T>` class

- custom collections, 235
- described, 132
- dictionary values, returning, 180
- performance, 260–261
- strings, populating with, 302–304
- validating, 244

literal strings, 201–202

loading types at run-time. *See* reflection

looking up keys, BCL, 154–156**loop**

- BCL generics, enabling, 236
- casting errors, handling, 30

M**mathematical functions**

- BCL collections, extending, 144–147
- bloat, 84
- values, comparing and returning greater, 83

maximum methods, 304**maximum value, returning, 224****memory footprint, .NET, 261–262****merging collections**

- algorithm, 318
- sets, 365
- two bags, 331

messages

- method for sending, 219
- remote access interface, 212

meta-programming, .NET rejection of, 245–246**methods**

- applying to individual, 16
- behavior, altering based on type, 219–220
- C++ generic classes, 268
- calling recursively, 256–257
- collection, searching, 168–171
- Collection<T> class, 137
- constraints, 87–88
- delegates with, 96–97, 106
- described, 83–87
- functions, applying similar, 48
- generic classes
 - arrays of type parameters, 65–66
 - described, 60–61
 - operator overloading, 66–67
 - overloading, 61–63
 - overriding, 63–65
- J#, calling, 287–288
- messages, sending, 219
- overloading, 88–90
- overriding, 92–94
- protected, 159–160
- type inference, 95
- type parameter names, 88
- type-safe database access example, 97–98
- uniqueness, 90–91

migrating from Java generics, 280**minimum methods, 304****modifying collection contents**

- adding, 309–311
- concatenating arrays, 312
- reversing order, 312
- rotating, 312–313

multiple constraint ambiguity

- avoiding, 237–238
- mixing classes and interfaces, 124–125
- problems, 123–124

multiple delegates, replacing, 225–227**multiple values**

- accessing, 343–345
- associating, 354–355
- extending, 345–346

N**name**

- collection classes, 235
- remote access clients, 213
- run-time loading
 - closed types, converting to open, 194–195
 - creating open and closed types, 189–193
 - described, 187
 - discerning types, 188–189
 - extending, 188
 - generic inheritance, 198–199
 - generic methods, 199–201
 - obfuscated environments, 201–202
 - open and closed types, 188
 - parameters and arguments, 195–198
 - type parameters, confusion caused by, 63, 228–231

nested

- C++ generic classes, 267–268
- generic classes, 67–69

.NET

- backward compatibility, 262
- blending generics, 44–45
- boxing, rejection of, 245
- code sharing, 248
- code specialization, 248
- constraints, 127–128
- declaration, validating at, 244
- described, 243
- exact run-time types, 254–256
- genetic specializations, creating on demand, 37
- hybrid model, 248
- IL representation, 246–247
- just-in-time, 249–253
- memory footprint, 261–262

.NET (continued)

- meta-programming, rejection of, 245–246
- NGen tool, 257–259
- performance, 259–261
- platform conformity, 245
- polymorphic recursion, support for, 256–257
- simplicity, 244

NGen tool, 257–259

node, linked lists, 160–161

non-type parameters, 44

null

- collection elements, 185
- keys, mapping to values, 157
- state, 241
- value, discerning, 74–76

number of items, reporting

- BCL collection
 - capacity, 165
 - dictionary, 179
 - general collection items, 136
 - keyed collection items, 159
 - linked list nodes, 160
 - list items, 162
 - queue, 175
 - read-only, 177
 - sorted list, 182
- Power Collections library
 - algorithm, 297
 - Bag<T>, 323
 - BigList<T>, 332
 - dictionary, 359
 - Set<T>, 362
- type parameters, 17

numbers, boxing to use as object, 5

O

obfuscation, literal string, 201–202

OO (object-oriented programming)

- parameters, passing pool of objects as, 11–12
- replacing with type parameters, 219

open constructed types, 15–16

open types

- constructed fields, 59–60
- creating, 189–193
- reflection, 188
- in shared code, just-in-time specialization, 251–252

operator overloading, 66–67

optional values, null state, 241

ordered collection, 350–354

ordered dictionary

- multiple values, associating with key, 354–355
- synchronized view, 355

ordered list

- BCL generics, 175–177
- double-ended
 - accessing and removing items, 339–341
 - adding items, 338–339
 - described, 337–338

ordered set, synchronized with subset, 358

organizing

- collection contents, 309–313
- generics, 218

overloading

- collection constructor, 135
- methods, 61–63, 88–90, 304
- types, 50, 269

overriding methods, 63–65, 92–94

P

Pair class, 358

parameterless constructors, providing, 238–239

parameters

- applying, 26–27
- arity, 17
- arrays, 65–66
- checking, 27–28
- comparing (EqualityComparer<T>), 157–158
- constraints
 - ambiguity, avoiding, 237–238
 - applying, 87–88
 - least restrictive, 236–237
- defined, 14–15
- generic classes, 47–49, 49–50
- generic methods within generic class, 230
- inheritance templates, 44
- maximum value, returning, 224
- methods, 86–87
- names
 - generics guidelines, 228–231
 - methods, 88
- new instances, declaring, 9
- object data type, replacing with, 9
- objects, passing as, 11–12
- outer classes, 68
- overloading generic methods, 88–90
- overuse, 233
- parameterless constructors, 238–239
- reflection, 195–198

parametric polymorphism, 13–14

parent and child node. See linked list

partitioning collection, 300, 301, 317–318

pattern, searching collection for, 300, 315

performance, .NET, 259–261

platform conformity, .NET, 245

polymorphic recursion, .NET support for, 256–257

populating

Bag<T> library, 324–325

collection state, 136–141

customer objects with orders, 23–24

generic types, generally, 9–10

large lists, managing (BigList<T> class), 333–334

object class, non-generics programming, 5

Set<T>, 364–365

Power Collections library

algorithms

class, 295

collection contents, inspecting, 302–304

comparing, 305–309, 321

copying collections, 305–307

listed, 296–301

modifying and arranging collection contents, 309–313

randomizing collections, 321

searching collections, 313–316

set operations, 318–321

sorting collections, 316–318

Bag<T>

accessing items, 326–327

construction and population, 324–325

described, 321–323

removing items, 327–329

set operations, performing, 329–332

base classes, 292–293

BinaryPredicate<T> delegate, 336

collections

implementing custom (CollectionBase<T> abstract class), 336–337

ordered view, 349–350

containers

CollectionBase<T> abstract class, 294–295

dictionary base classes, 295

ListBase<T> abstract class, 293–294

delegates, 296

derived dictionary types, 341

described, 291

dictionary

multiple values, accessing with key, 343–345

multiple values, extending, 345–346

subset, synchronized view of, 354

double-ended queue

accessing and removing items, 339–341

adding items, 338–339

described, 337–338

generic structures, 295–296

keyed random access and ordered collection, 350–354

large lists, managing (BigList<T> class)

accessing contents, 334–335

construction and population, 333–334

described, 332–333

removing items, 335–336

list-based derived types, 342–343

ordered dictionary

multiple values, associating with key, 354–355

synchronized view, 355

ordered set, 358

Pair class, 358

read-only collection

described, 358–359

with dictionary key, 359–360

dictionary states, locking, 361–362

lists, 360–361

Set<T> class, 362–366

sorted version of Bag<T> class

adding and removing items, 347–348

described, 346–347

range operations, 348–349

sorted view of set, 355–357

Triple class, 366

programming language, C#

accessibility rules, 72–73

angle bracket syntax, 131

anonymous methods, 109

class constraints, 125

delegates, supplying, 168

method signature, 63

multiple constraints, 122–123

nullable types, declaring, 76

overriding generic methods, 93, 94

parameter names, 228

ref types, using type parameters for, 220–221

programming language, C++. See also .NET

blurring lines, 264

code bloat, 42

compatibility, 264

constraints, 42, 271–273

generic classes

consuming, 265–266

default types, 268

described, 264–265

programming language, C++ (continued)

- inheritance, 266–267
- methods, 268
- nested, 267–268
- generic delegates, 273–274
- generic interfaces, 270
- generic methods, 268–270
- STL.NET, 277–278
- templates
 - generics versus, 263
 - least common denominator programming, 20
 - mixing with generics, 274–277

programming language, J#

- arrays of generic types, 289–290
- constraints, 285–287
- cross-language support, 281–283
- generic delegates, calling, 289
- generic types, consuming, 280–281
- interfaces, 283–285
- limitations, 279
- methods, calling, 287–288
- migrating from Java generics, 280
- type arguments, 288

properties

- generic classes, 77–80
- protected, 159–160

protected members, 55–56

protected properties and methods, 159–160

Pyramid Manager sample program

- generics, 7–10
- non-generic coding, problems with, 2–5
- type-safe version, 5–7

Q

queue

- BCL generics, 175–177
- double-ended
 - accessing and removing items, 339–341
 - adding items, 338–339
 - described, 337–338

R

random access keys, 350–354

randomizing collections, 321

range operations

- ordered dictionary, 353
- sorted version, 348–349

readability, balancing with expressiveness, 227–228

read-only collection

- BCL
 - copy, returning, 162
 - generally, 177–179
- Power Collections
 - described, 300, 358–359
 - with dictionary key, 359–360
 - dictionary states, locking, 361–362
 - lists, 360–361

recently deleted history, recording, 144–147

refactoring, candidate methods for, 223–225

reference, type not, 44

reflection

- closed types, converting to open, 194–195
- creating open and closed types, 189–193
- described, 187
- discerning types, 188–189
- extending, 188
- generic inheritance, 198–199
- generic methods, 199–201
- obfuscated environments, 201–202
- open and closed types, 188
- parameters and arguments, 195–198

relationships, tracking salespeople in pyramid scheme

- generics, 7–10
- non-generic coding, problems with, 2–5
- type-safe version, 5–7

remote access

- client calling code, 213–215
- console application code, 212–213
- interface, 211–212

removing items

- algorithm, 313
- Bag<T> class
 - duplicates, 323
 - library, 327–329
 - method, 321
 - sorted version, 347–348

BCL collections

- dictionary, 179
- generally, 141–142
- keyed objects, 159
- keys mapped to values, 157
- list, 163
- listed list nodes, 160
- from queue, 175
- sorted list, 182
- stack, 183
- BigList<T>, 332, 333

dictionary, 345
 double-ended queue, 339–341
 internally implemented classes, 167–168
 large lists, managing (`BigList<T>` class), 335–336
 object at specified index, 136
 read-only collection, 300
`Set<T>`, 362, 363

reordering collection contents, 300

representative objects, 325

retired employees, finding with parameters, 47–48

retrieving

keys, mapping to values, 156–157
 objects from database, 97–98

reversing order

`BigList<T>`, 333
 collection contents, 312
 read-only collection, 300

rotating collection contents, 300, 312–313

rules violations

benefits, 19–20
 casting, 29–31
 casting, caution against, 5
 clarity versus, 32–33
 client, shielding, 223
 database access example, 97–98
 delegates
 action on object, 106–107
 anonymous methods, 109
 BCL generics, 134
 choosing, 107
 constraints, applying, 108–109
 criteria, item meeting set, 107
 described, 99–102
 event handlers, 105–106
 generics and, 102–105
 methods, 96–97, 227
 with methods, 106
 Power Collections library, 296
 supplying, 168
 two objects, comparing, 107
 type coercion, 107–108
 type, converting, 107
 generality versus, 219
 interface, 31
 least common denominator programming, 20
 linked list wrapper, 161
 sample, 21–29
 serialization, 206–207

run-time instantiation, 37–38, 40–42

run-time, names loaded at. See reflection

S

searching

`Collection<T>` class, 136
 internally implemented BCL classes, 168–171
 Power Collections algorithms, 313–316

serialization

custom, 204–208
 mechanics, 202–204
 with web services, 208–210

server, remote access

application code, 212–213
 client calls, 213–215

set operations

`Bag<T>` library, 329–332
 Power Collections algorithms, 318–321

`Set<T>` class

adding and editing items, 362–363
 populating, 364–365
 using, 366

sharing

just-in-time specialization, 250–251
 .NET
 code sharing, 248
 code specialization, 248
 hybrid model, 248
 just-in-time, 249–253
 NGen, 257

shuffling collection contents, 300

signatures

delegate, 226
 dictionary class, 344–345
 methods, 89–90, 94
 type parameters, 63, 66

simplicity, .NET, 244

size

list
 pre-allocating, 164–165
 setting, 163
 queue, 176
 stack, 184

sorted version of `Bag<T>` class

adding and removing items, 347–348
 described, 346–347
 range operations, 348–349

sorted view of Power Collections set, 355–357

sorting

BCL generics
 dictionary based on key, 182–183
 items, internally implemented classes, 173–175
 list items, 163

sorting (continued)

- objects of any type, 220
- Power Collections
 - algorithms, 301, 316–318
 - BigList<T>, 333

specialization

- BCL custom classes, 235
- .NET
 - code sharing, 248
 - code specialization, 248
 - hybrid model, 248
 - just-in-time, 249–253
 - templates, 43–44

specific order, dictionary

- items, adding and editing, 179
- keys, 180–181
- sorting, 182–183

stacked class

- BCL generics, 183–184
- run-time instantiation, 37–38

Standard Template Library (STL), 292

static constructors, 51

static data members, 239–240

static fields, 57–59

static methods

- generic and non-generic, caution against mixing, 234
- PCL library, 301–302

STL (Standard Template Library), 292

STL.NET, 277–278

stock price example

- boxing, 121
- clients feeding new prices, 120–121
- interface and structure, 119–120
- updates, 121–122

string

- CollectionBase<T>, 337
- dictionary
 - filling, 345
 - outputting, 362
- method, 301
- populating List<T>, 302–304
- sorting, 316–317
- Triple class, 366

structs, 80–81

subclasses

- C++ generic classes, 266–267
- generally, 52–55
- generic constraints, 126–127

- polymorphism, achieving, 13
- protected members, 55–56
- reflection, 198–199
- type, specifying, 28

subset, synchronized view

- Bag<T>, 322, 330–331
- BigList<T>, 333
- method, 299
- ordered set, 358
- read-only dictionaries, 361
- Set<T>, 363

swapping method, 224

synchronized view, Power Collections ordered dictionary, 355

System.Collections.ObjectModel

namespace, 133

System.type, **replacing with type parameters, 219–220**

T

templates

- assemblies and type equivalence, 42–43
- C++
 - advantages, 263
 - least common denominator programming, 20
 - mixing with generics, 274–277
- code bloat, 42
- compile-time instantiation, 36–37, 40, 43
- constraints, objection to, 127
- debugging, 44
- generics versus, 35, 38–42
- non-type parameters, 44
- specialization, 43–44
- type parameter inheritance, 44

testing equality

- BCL generics, 157–158, 184–185
- Power Collections, 297
- queue objects, 175
- two objects, 132

top of stack, returning item to, 184

transforming BCL contents, 171–173

tree structure, 3–4

Triple class, 366

two objects, comparing, 107

type

- casting, 255–256
- coercion, 107–108

- comparisons, 241
- compiler generating, 43
- conversion, 67, 107
- equivalence, 42–43
- generic constraints, 117–118
- info, 76–77
- instantiation, 16–17
- neither reference nor value parameter, 44
- parameterizing, 13–14
- safe approach, 5–7
- template placeholders, 36

type arguments

- alternative template implementation, 43
- constructed types, caution against, 232–233
- described, 15
- J#, 288

type inference

- generics guidelines, 233–234
- methods, 95

type parameter

- arity, 17
- arrays, 65–66
- comparing (`EqualityComparer<T>`), 157–158
- constraints
 - ambiguity, avoiding, 237–238
 - applying, 87–88
 - least restrictive, 236–237
- defined, 14–15
- generic classes, 49–50
- generic methods within generic class, 230
- inheritance templates, 44
- methods, 86–87
- names
 - generics guidelines, 228–231
 - methods, 88
- new instances, declaring, 9
- object data type, replacing with, 9
- outer classes, 68
- overloading generic methods, 88–90
- overuse, 233
- parameterless constructors, 238–239

type safety

- benefits, 19–20
- casting, 29–31
- casting, caution against, 5
- clarity versus, 32–33
- client, shielding, 223
- database access example, 97–98

- delegates
 - action on object, 106–107
 - anonymous methods, 109
 - BCL generics, 134
 - choosing, 107
 - constraints, applying, 108–109
 - criteria, item meeting set, 107
 - described, 99–102
 - event handlers, 105–106
 - generics and, 102–105
 - methods, 96–97, 227
 - with methods, 106
 - Power Collections library, 296
 - supplying, 168
 - two objects, comparing, 107
 - type coercion, 107–108
 - type, converting, 107
- generality versus, 219
- interface, 31
- least common denominator programming, 20
- linked list wrapper, 161
- sample, 21–29
- serialization, 206–207

U

union, two sets, 301, 363

uniqueness, method, 90–91

updating

- collections, 136–141
- internally implemented classes, 165–166
- keys, mapping to values, 152–153

user interface

- BCL generics, 130–132
- C++ class template declaration, 274–277
- classes versus, 240–241
- constraints, 123–124
- generic classes, 81–82
- generic constraints, 115
- generics guidelines, 227
- J#
 - basic, 283–285
 - constraints, 285–287
- remote access, 211–212
- type parameters in, 62–63
- type safety
 - described, 31
 - preserving, 25–26

V

value parameter

- blocking, 119–122
- linked list node, 161
- type not, 44

VB (Visual Basic)

- accessibility rules, 72–73
- class constraints, 125
- delegates, supplying, 168
- method signature, 63
- multiple constraints, 122
- operator overloading, enhanced, 66

- overriding methods, 93, 94
- static data members, 239
- types, declaring, 221

verbosity, 32

W

web services, 208–210

wrappers, generic, 301

X

XML (eXtensible Markup Language), 210